MSD Script

Generated by Doxygen 1.10.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Add Class Reference

Inheritance diagram for Add:

class_add-eps-converted-to.pdf

Public Member Functions

- Add (Expr ∗lhs, Expr ∗rhs)

  Constructor for the Add class. Creates the Add object with left and right expressions.
- bool equals (Expr ∗e)

  Implementation of the equals function for Add.
- int interp ()

  the interp() function for Add class.
- bool has_variable ()

  Checks if the expression contains any variables.
- Expr ∗ subst (string varName, Expr ∗replacement)

  Substitutes a variable within the expression with another expression.
- virtual void print (ostream &os)

  Prints the Add expression to an output stream.
- void pretty_print_at (ostream &ot, precedence_t prec)

  Pretty prints the Add expression with correct precedence handling.

Public Member Functions inherited from Expr

- string **to_string** ()
- void **pretty_print** (ostream &ostream)
- string **to_pretty_string** ()

Public Attributes

- Expr ∗ **lhs**
- Expr ∗ **rhs**

4.1.1 Constructor & Destructor Documentation

4.1.1.1 Add()

```
Add::Add (
            Expr * lhs,
            Expr * rhs )
```

Constructor for the Add class. Creates the Add object with left and right expressions.

Parameters

lhs	The left expression.
rhs	The right expression.

4.1.2 Member Function Documentation

4.1.2.1 equals()

```
bool Add::equals (
            Expr * e )  [virtual]
```

Implementation of the equals function for Add.

Parameters

e	the expression you compare.

Returns

false if add is a null pointer, true otherwise. Verifies the current Var object is equal to a different expression.

Implements Expr.

4.1.2.2 has_variable()

```
bool Add::has_variable ( )  [virtual]
```

Checks if the expression contains any variables.

Returns

True if either lhs or rhs contains a variable, false otherwise.

Implements Expr.

4.1.2.3 interp()

```
int Add::interp ( )  [virtual]
```

the interp() function for Add class.

Returns

lefthand side and righthand side with the Interp() method.

Implements Expr.

4.1.2.4 pretty_print_at()

```
void Add::pretty_print_at (
            ostream & ot,
            precedence_t prec )  [virtual]
```

Pretty prints the Add expression with correct precedence handling.

Parameters

o	The output stream to print to.
prec	The precedence level of the expression's context.

Reimplemented from Expr.

4.1.2.5 print()

```
void Add::print (
            ostream & ostream )  [virtual]
```

Prints the Add expression to an output stream.

Parameters

ostream	The output stream to print to.

Implements Expr.

4.1.2.6 subst()

```
Expr * Add::subst (
            string varName,
            Expr * replacement )  [virtual]
```

Substitutes a variable within the expression with another expression.

Parameters

varName	The name of the variable to be substituted.
replacement	The expression to substitute in place of the variable.

Returns

A new Add expression with the variable substituted.

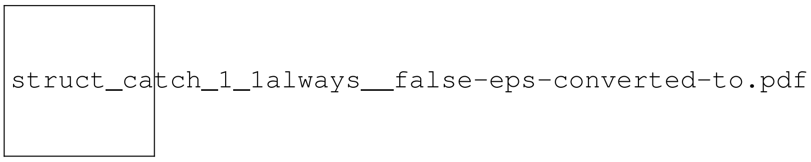
Implements Expr.

The documentation for this class was generated from the following files:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h
- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp

4.2 Catch::always_false< T > Struct Template Reference

Inheritance diagram for Catch::always_false< T >:



The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.3 Catch::Detail::Approx Class Reference

Public Member Functions

- **Approx** (double value)
- Approx **operator-** () const
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  Approx **operator()** (T const &value) const
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  Approx (T const &value)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  Approx & **epsilon** (T const &newEpsilon)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  Approx & **margin** (T const &newMargin)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  Approx & **scale** (T const &newScale)
- std::string **toString** () const

Static Public Member Functions

- static Approx **custom** ()

Friends

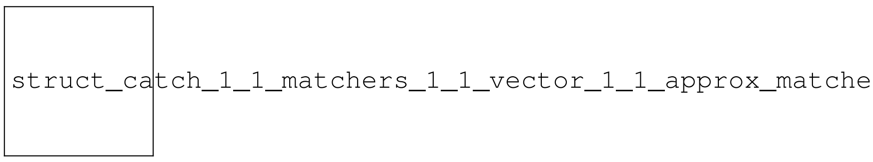
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator==** (const T &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator==** (Approx const &lhs, const T &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator!=** (T const &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator!=** (Approx const &lhs, T const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator<=** (T const &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator<=** (Approx const &lhs, T const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator>=** (T const &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  bool **operator>=** (Approx const &lhs, T const &rhs)

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.4 Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >:



Public Member Functions

- **ApproxMatcher** (std::vector< T, AllocComp > const &comparator)
- bool **match** (std::vector< T, AllocMatch > const &v) const override
- std::string describe () const override
- template<typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  ApproxMatcher & **epsilon** (T const &newEpsilon)
- template<typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  ApproxMatcher & **margin** (T const &newMargin)
- template<typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
  ApproxMatcher & **scale** (T const &newScale)

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Public Attributes

- std::vector< T, AllocComp > const & **m_comparator**
- Catch::Detail::Approx **approx** = Catch::Detail::Approx::custom()

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.4.1 Member Function Documentation

4.4.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::ApproxMatcher< T, AllocComp, AllocMatch >::describe ( )
const  [inline], [override], [virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.5 Catch::Generators::as< T > Struct Template Reference

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.6 Catch::AssertionHandler Class Reference

Public Member Functions

- **AssertionHandler** (StringRef const &macroName, SourceLineInfo const &lineInfo, StringRef captured↩ Expression, ResultDisposition::Flags resultDisposition)
- template<typename T >
  void **handleExpr** (ExprLhs< T > const &expr)
- void **handleExpr** (ITransientExpression const &expr)
- void **handleMessage** (ResultWas::OfType resultType, StringRef const &message)
- void **handleExceptionThrownAsExpected** ()
- void **handleUnexpectedExceptionNotThrown** ()
- void **handleExceptionNotThrownAsExpected** ()
- void **handleThrowingCallSkipped** ()
- void **handleUnexpectedInflightException** ()
- void **complete** ()
- void **setCompleted** ()
- auto **allowThrows** () const -> bool

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.7 Catch::AssertionInfo Struct Reference

Public Attributes

- StringRef **macroName**
- SourceLineInfo **lineInfo**
- StringRef **capturedExpression**
- ResultDisposition::Flags **resultDisposition**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.8 Catch::AssertionReaction Struct Reference

Public Attributes

- bool **shouldDebugBreak** = false
- bool **shouldThrow** = false

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.9 Catch::AutoReg Struct Reference

Inheritance diagram for Catch::AutoReg:

struct_catch_1_1_auto_reg-eps-converted-to.pdf

Public Member Functions

- **AutoReg** (ITestInvoker ∗invoker, SourceLineInfo const &lineInfo, StringRef const &classOrMethod, NameAndTags const &nameAndTags) noexcept

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.10 Catch::BinaryExpr< LhsT, RhsT > Class Template Reference

Inheritance diagram for Catch::BinaryExpr< LhsT, RhsT >:

class_catch_1_1_binary_expr-eps-converted-to.pdf

Public Member Functions

- **BinaryExpr** (bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs)
- template<typename T >
  auto **operator&&** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >
  auto **operator**|| (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >
  auto **operator==** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >
  auto **operator!=** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >
  auto **operator**> (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >
  auto **operator**< (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >
  auto **operator>=** (T) const -> BinaryExpr< LhsT, RhsT const & > const
- template<typename T >
  auto **operator<=** (T) const -> BinaryExpr< LhsT, RhsT const & > const

Public Member Functions inherited from Catch::ITransientExpression

- auto **isBinaryExpression** () const -> bool
- auto **getResult** () const -> bool
- **ITransientExpression** (bool isBinaryExpression, bool result)

Additional Inherited Members

Public Attributes inherited from Catch::ITransientExpression

- bool **m_isBinaryExpression**
- bool **m_result**

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.11 Catch::Capturer Class Reference

Public Member Functions

- **Capturer** (StringRef macroName, SourceLineInfo const &lineInfo, ResultWas::OfType resultType, StringRef names)
- void **captureValue** (size_t index, std::string const &value)
- template<typename T >
  void **captureValues** (size_t index, T const &value)
- template<typename T , typename... Ts>
  void **captureValues** (size_t index, T const &value, Ts const &... values)

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.12 Catch::Matchers::StdString::CasedString Struct Reference

Public Member Functions

- **CasedString** (std::string const &str, CaseSensitive::Choice caseSensitivity)
- std::string **adjustString** (std::string const &str) const
- std::string **caseSensitivitySuffix** () const

Public Attributes

- CaseSensitive::Choice **m_caseSensitivity**
- std::string **m_str**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.13 Catch::CaseSensitive Struct Reference

Public Types

- enum **Choice** { **Yes** , **No** }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

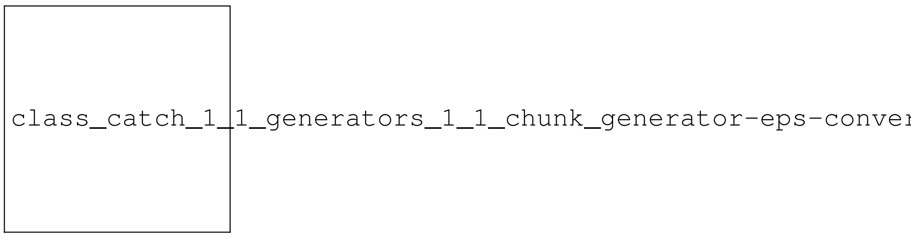
4.14 Catch_global_namespace_dummy Struct Reference

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.15 Catch::Generators::ChunkGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::ChunkGenerator< T >:



Public Member Functions

- **ChunkGenerator** (size_t size, GeneratorWrapper< T > generator)
- std::vector< T > const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< std::vector< T > >

- using **type**

4.15.1 Member Function Documentation

4.15.1.1 get()

```
template<typename T >
std::vector< T > const & Catch::Generators::ChunkGenerator< T >::get ( ) const  [inline],
[override], [virtual]
```

Implements Catch::Generators::IGenerator< std::vector< T > >.

4.15.1.2 next()

```
template<typename T >
bool Catch::Generators::ChunkGenerator< T >::next ( )  [inline], [override], [virtual]
```

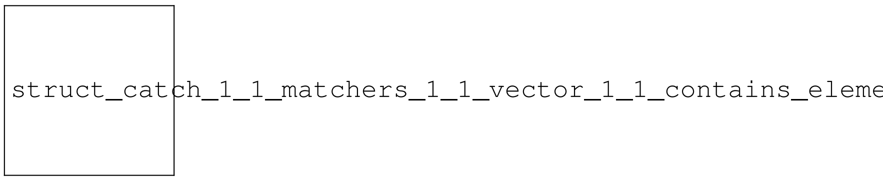
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.16 Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >:



Public Member Functions

- **ContainsElementMatcher** (T const &comparator)
- bool **match** (std::vector< T, Alloc > const &v) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Public Attributes

- T const & **m_comparator**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.16.1 Member Function Documentation

4.16.1.1 describe()

```
template<typename T , typename Alloc >
std::string Catch::Matchers::Vector::ContainsElementMatcher< T, Alloc >::describe ( ) const
[inline], [override], [virtual]
```

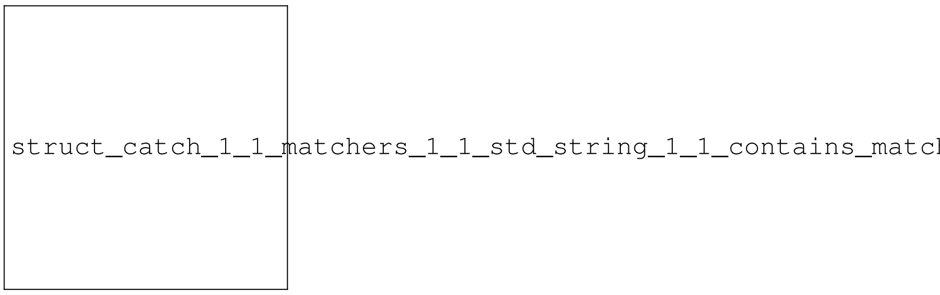
Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.17 Catch::Matchers::StdString::ContainsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::ContainsMatcher:



Public Member Functions

- **ContainsMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Public Member Functions inherited from Catch::Matchers::StdString::StringMatcherBase

- **StringMatcherBase** (std::string const &operation, CasedString const &comparator)
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator**|| (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Public Attributes inherited from Catch::Matchers::StdString::StringMatcherBase

- CasedString **m_comparator**
- std::string **m_operation**

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

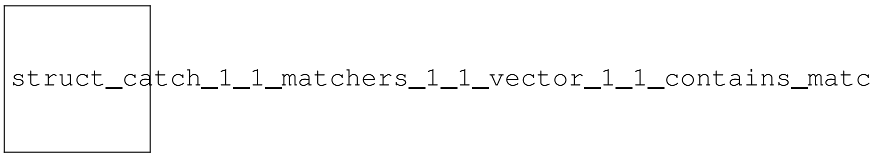
- std::string **m_cachedToString**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.18 Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >:



Public Member Functions

- **ContainsMatcher** (std::vector< T, AllocComp > const &comparator)
- bool **match** (std::vector< T, AllocMatch > const &v) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Public Attributes

- std::vector< T, AllocComp > const & **m_comparator**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.18.1 Member Function Documentation

4.18.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::ContainsMatcher< T, AllocComp, AllocMatch >::describe ( )
const  [inline], [override], [virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.19 Catch::Counts Struct Reference

Public Member Functions

- Counts **operator-** (Counts const &other) const
- Counts & **operator+=** (Counts const &other)
- std::size_t **total** () const
- bool **allPassed** () const
- bool **allOk** () const

Public Attributes

- std::size_t **passed** = 0
- std::size_t **failed** = 0
- std::size_t **failedButOk** = 0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.20 Catch::Decomposer Struct Reference

Public Member Functions

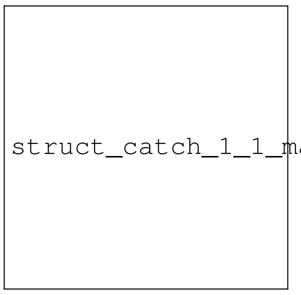
- template<typename T >
  auto **operator<=** (T const &lhs) -> ExprLhs< T const & >
- auto **operator<=** (bool value) -> ExprLhs< bool >

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.21 Catch::Matchers::StdString::EndsWithMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::EndsWithMatcher:



Public Member Functions

- **EndsWithMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Public Member Functions inherited from Catch::Matchers::StdString::StringMatcherBase

- **StringMatcherBase** (std::string const &operation, CasedString const &comparator)
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator**|| (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Public Attributes inherited from Catch::Matchers::StdString::StringMatcherBase

- CasedString **m_comparator**
- std::string **m_operation**

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.22 Catch::Detail::EnumInfo Struct Reference

Public Member Functions

- StringRef **lookup** (int value) const

Public Attributes

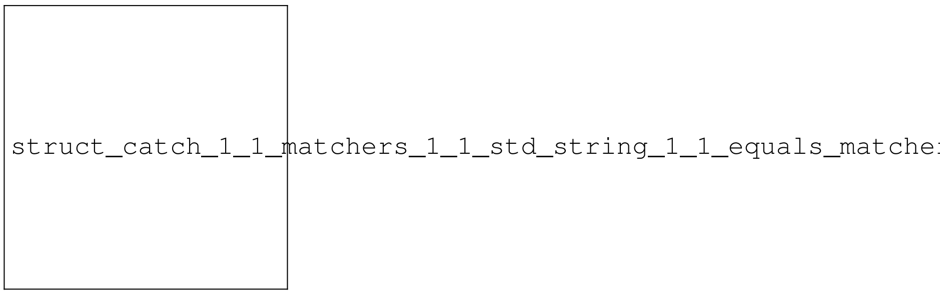
- StringRef **m_name**
- std::vector< std::pair< int, StringRef > > **m_values**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.23 Catch::Matchers::StdString::EqualsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::EqualsMatcher:



Public Member Functions

- **EqualsMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Public Member Functions inherited from Catch::Matchers::StdString::StringMatcherBase

- **StringMatcherBase** (std::string const &operation, CasedString const &comparator)
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Public Attributes inherited from Catch::Matchers::StdString::StringMatcherBase

- CasedString **m_comparator**
- std::string **m_operation**

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

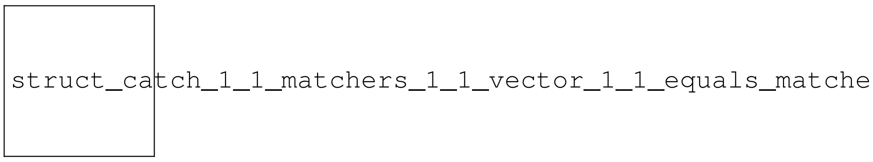
- std::string **m_cachedToString**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.24 Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >:



Public Member Functions

- **EqualsMatcher** (std::vector< T, AllocComp > const &comparator)
- bool **match** (std::vector< T, AllocMatch > const &v) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Public Attributes

- std::vector< T, AllocComp > const & **m_comparator**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.24.1 Member Function Documentation

4.24.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::EqualsMatcher< T, AllocComp, AllocMatch >::describe ( )
const  [inline], [override], [virtual]
```

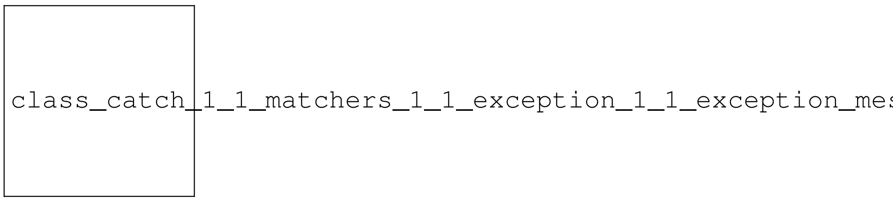
Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.25 Catch::Matchers::Exception::ExceptionMessageMatcher Class Reference

Inheritance diagram for Catch::Matchers::Exception::ExceptionMessageMatcher:



Public Member Functions

- **ExceptionMessageMatcher** (std::string const &message)
- bool **match** (std::exception const &ex) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator**|| (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.25.1 Member Function Documentation

4.25.1.1 describe()

```
std::string Catch::Matchers::Exception::ExceptionMessageMatcher::describe ( ) const  [override],
[virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.26 Catch::ExceptionTranslatorRegistrar Class Reference

Public Member Functions

- template< typename T >
  ExceptionTranslatorRegistrar (std::string(∗translateFunction)(T &))

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.27 Expr Class Reference

Inheritance diagram for Expr:

class_expr-eps-converted-to.pdf

Public Member Functions

- virtual bool equals (Expr ∗e)=0
- virtual int interp ()=0
- virtual bool has_variable ()=0
- virtual Expr ∗ subst (string varName, Expr ∗replacement)=0
- virtual void print (ostream &os)=0
- string **to_string** ()
- void **pretty_print** (ostream &ostream)
- virtual void pretty_print_at (ostream &ot, precedence_t prec)
- string **to_pretty_string** ()

4.27.1 Member Function Documentation

4.27.1.1 equals()

```
virtual bool Expr::equals (
             Expr * e )  [pure virtual]
```

Implemented in Num, Var, Add, and Mult.

4.27.1.2 has_variable()

```
virtual bool Expr::has_variable ( )  [pure virtual]
```

Implemented in Num, Var, Add, and Mult.

4.27.1.3 interp()

```
virtual int Expr::interp ( )  [pure virtual]
```

Implemented in Num, Var, Add, and Mult.

4.27.1.4 pretty_print_at()

```
void Expr::pretty_print_at (
             ostream & ot,
             precedence_t prec )  [virtual]
```

Reimplemented in Add, and Mult.

4.27.1.5 print()

```
virtual void Expr::print (
            ostream & os )  [pure virtual]
```

Implemented in Num, Var, Add, and Mult.

4.27.1.6 subst()

```
virtual Expr * Expr::subst (
            string varName,
            Expr * replacement )  [pure virtual]
```

Implemented in Num, Var, Add, and Mult.

The documentation for this class was generated from the following files:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h
- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp

4.28 Catch::ExprLhs< LhsT > Class Template Reference

Public Member Functions

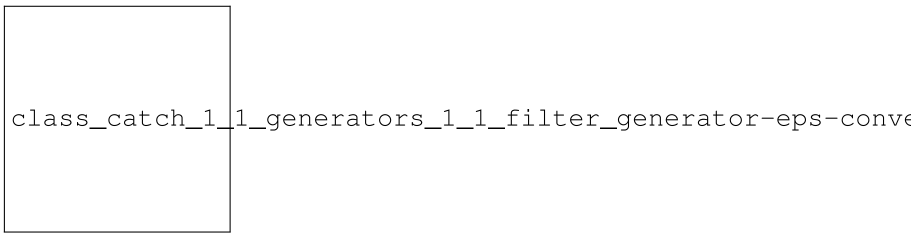
- **ExprLhs** (LhsT lhs)
- template<typename RhsT >
  auto **operator==** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- auto **operator==** (bool rhs) -> BinaryExpr< LhsT, bool > const
- template<typename RhsT >
  auto **operator!=** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- auto **operator!=** (bool rhs) -> BinaryExpr< LhsT, bool > const
- template<typename RhsT >
  auto **operator>** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator<** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator>=** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator<=** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator|** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator&** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator^** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator&&** (RhsT const &) -> BinaryExpr< LhsT, RhsT const & > const
- template<typename RhsT >
  auto **operator||** (RhsT const &) -> BinaryExpr< LhsT, RhsT const & > const
- auto **makeUnaryExpr** () const -> UnaryExpr< LhsT >

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.29 Catch::Generators::FilterGenerator< T, Predicate > Class Template Reference

Inheritance diagram for Catch::Generators::FilterGenerator< T, Predicate >:



Public Member Functions

- template<typename P = Predicate>
  FilterGenerator (P &&pred, GeneratorWrapper< T > &&generator)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.29.1 Member Function Documentation

4.29.1.1 get()

```
template<typename T , typename Predicate >
T const & Catch::Generators::FilterGenerator< T, Predicate >::get ( ) const  [inline], [override],
[virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.29.1.2 next()

```
template<typename T , typename Predicate >
bool Catch::Generators::FilterGenerator< T, Predicate >::next ( )  [inline], [override], [virtual]
```

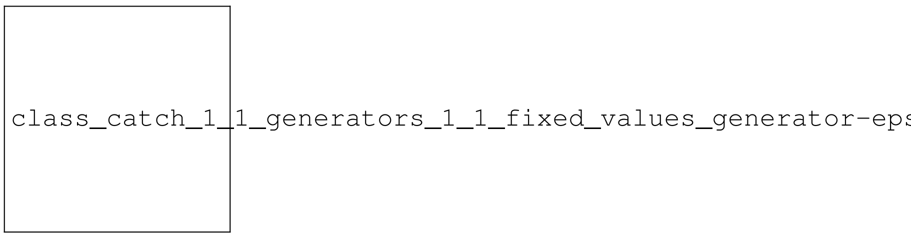
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.30 Catch::Generators::FixedValuesGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::FixedValuesGenerator< T >:



Public Member Functions

- **FixedValuesGenerator** (std::initializer_list< T > values)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.30.1 Member Function Documentation

4.30.1.1 get()

```
template<typename T >
T const & Catch::Generators::FixedValuesGenerator< T >::get ( ) const  [inline], [override],
[virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.30.1.2 next()

```
template<typename T >
bool Catch::Generators::FixedValuesGenerator< T >::next ( )  [inline], [override], [virtual]
```

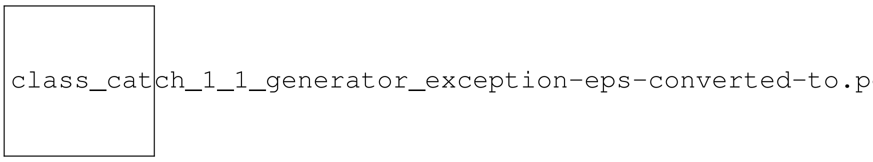
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.31 Catch::GeneratorException Class Reference

Inheritance diagram for Catch::GeneratorException:



Public Member Functions

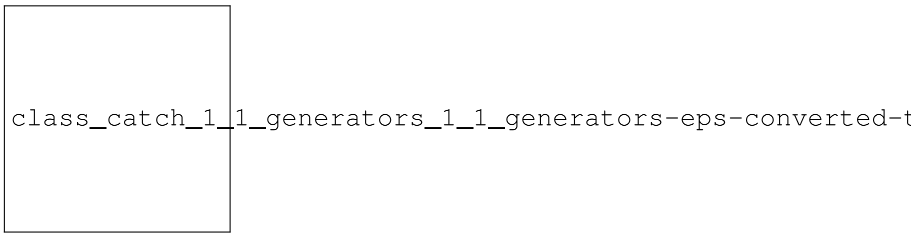
- **GeneratorException** (const char ∗msg)
- const char ∗ **what** () const noexcept override final

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.32 Catch::Generators::Generators< T > Class Template Reference

Inheritance diagram for Catch::Generators::Generators< T >:



Public Member Functions

- template<typename... Gs>
  Generators (Gs &&... moreGenerators)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.32.1 Member Function Documentation

4.32.1.1 get()

```
template<typename T >
T const & Catch::Generators::Generators< T >::get ( ) const  [inline], [override], [virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.32.1.2 next()

```
template<typename T >
bool Catch::Generators::Generators< T >::next ( )  [inline], [override], [virtual]
```

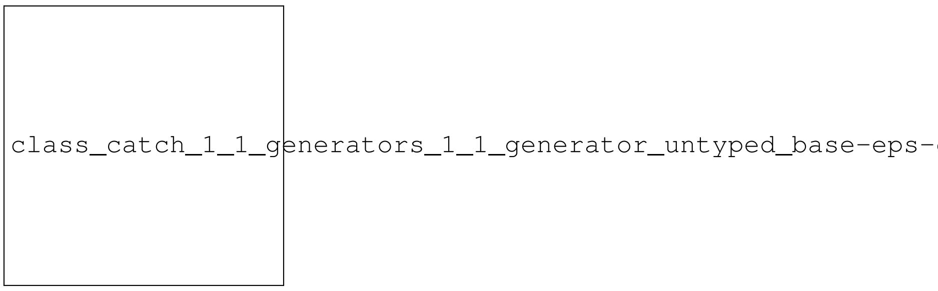
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.33 Catch::Generators::GeneratorUntypedBase Class Reference

Inheritance diagram for Catch::Generators::GeneratorUntypedBase:



Public Member Functions

- virtual bool **next** ()=0

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.34 Catch::Generators::GeneratorWrapper< T > Class Template Reference

Public Member Functions

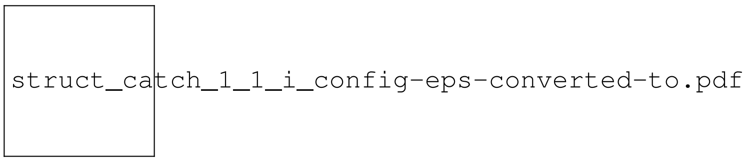
- **GeneratorWrapper** (std::unique_ptr< IGenerator< T > > generator)
- T const & **get** () const
- bool **next** ()

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.35 Catch::IConfig Struct Reference

Inheritance diagram for Catch::IConfig:



Public Member Functions

- virtual bool **allowThrows** () const =0
- virtual std::ostream & **stream** () const =0
- virtual std::string **name** () const =0
- virtual bool **includeSuccessfulResults** () const =0
- virtual bool **shouldDebugBreak** () const =0
- virtual bool **warnAboutMissingAssertions** () const =0
- virtual bool **warnAboutNoTests** () const =0
- virtual int **abortAfter** () const =0
- virtual bool **showInvisibles** () const =0
- virtual ShowDurations::OrNot **showDurations** () const =0
- virtual double **minDuration** () const =0
- virtual TestSpec const & **testSpec** () const =0
- virtual bool **hasTestFilters** () const =0
- virtual std::vector< std::string > const & **getTestsOrTags** () const =0
- virtual RunTests::InWhatOrder **runOrder** () const =0
- virtual unsigned int **rngSeed** () const =0
- virtual UseColour::YesOrNo **useColour** () const =0
- virtual std::vector< std::string > const & **getSectionsToRun** () const =0
- virtual Verbosity **verbosity** () const =0
- virtual bool **benchmarkNoAnalysis** () const =0
- virtual int **benchmarkSamples** () const =0
- virtual double **benchmarkConfidenceInterval** () const =0
- virtual unsigned int **benchmarkResamples** () const =0
- virtual std::chrono::milliseconds **benchmarkWarmupTime** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.36 Catch::IContext Struct Reference

Inheritance diagram for Catch::IContext:

struct_catch_1_1_i_context-eps-converted-to.pdf

Public Member Functions

- virtual IResultCapture ∗ **getResultCapture** ()=0
- virtual IRunner ∗ **getRunner** ()=0
- virtual IConfigPtr const & **getConfig** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.37 Catch::IExceptionTranslator Struct Reference

Public Member Functions

- virtual std::string **translate** (ExceptionTranslators::const_iterator it, ExceptionTranslators::const_iterator itEnd) const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.38 Catch::IExceptionTranslatorRegistry Struct Reference

Public Member Functions

- virtual std::string **translateActiveException** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.39 Catch::Generators::IGenerator< T > Struct Template Reference

Inheritance diagram for Catch::Generators::IGenerator< T >:

struct_catch_1_1_generators_1_1_i_generator-eps-converted-to.pdf

Public Types

- using **type** = T

Public Member Functions

- virtual T const & **get** () const =0

Public Member Functions inherited from Catch::Generators::GeneratorUntypedBase

- virtual bool **next** ()=0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.40 Catch::IGeneratorTracker Struct Reference

Public Member Functions

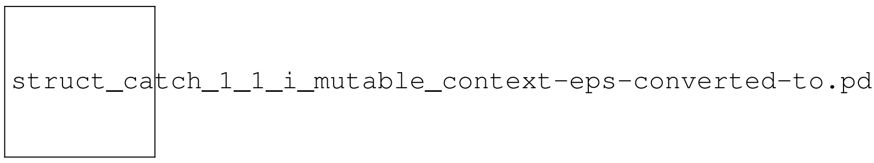
- virtual auto **hasGenerator** () const -> bool=0
- virtual auto **getGenerator** () const -> Generators::GeneratorBasePtr const &=0
- virtual void **setGenerator** (Generators::GeneratorBasePtr &&generator)=0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.41 Catch::IMutableContext Struct Reference

Inheritance diagram for Catch::IMutableContext:



Public Member Functions

- virtual void **setResultCapture** (IResultCapture ∗resultCapture)=0
- virtual void **setRunner** (IRunner ∗runner)=0
- virtual void **setConfig** (IConfigPtr const &config)=0

Public Member Functions inherited from Catch::IContext

- virtual IResultCapture ∗ **getResultCapture** ()=0
- virtual IRunner ∗ **getRunner** ()=0
- virtual IConfigPtr const & **getConfig** () const =0

Friends

- IMutableContext & **getCurrentMutableContext** ()
- void **cleanUpContext** ()

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.42 Catch::IMutableEnumValuesRegistry Struct Reference

Public Member Functions

- virtual Detail::EnumInfo const & **registerEnum** (StringRef enumName, StringRef allEnums, std::vector< int > const &values)=0
- template<typename E >
  Detail::EnumInfo const & **registerEnum** (StringRef enumName, StringRef allEnums, std::initializer_list< E > values)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.43 Catch::IMutableRegistryHub Struct Reference

Public Member Functions

- virtual void **registerReporter** (std::string const &name, IReporterFactoryPtr const &factory)=0
- virtual void **registerListener** (IReporterFactoryPtr const &factory)=0
- virtual void **registerTest** (TestCase const &testInfo)=0
- virtual void **registerTranslator** (const IExceptionTranslator ∗translator)=0
- virtual void **registerTagAlias** (std::string const &alias, std::string const &tag, SourceLineInfo const &line↩
Info)=0
- virtual void **registerStartupException** () noexcept=0
- virtual IMutableEnumValuesRegistry & **getMutableEnumValuesRegistry** ()=0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.44 Catch::IRegistryHub Struct Reference

Public Member Functions

- virtual IReporterRegistry const & **getReporterRegistry** () const =0
- virtual ITestCaseRegistry const & **getTestCaseRegistry** () const =0
- virtual ITagAliasRegistry const & **getTagAliasRegistry** () const =0
- virtual IExceptionTranslatorRegistry const & **getExceptionTranslatorRegistry** () const =0
- virtual StartupExceptionRegistry const & **getStartupExceptionRegistry** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.45 Catch::IResultCapture Struct Reference

Public Member Functions

- virtual bool **sectionStarted** (SectionInfo const &sectionInfo, Counts &assertions)=0
- virtual void **sectionEnded** (SectionEndInfo const &endInfo)=0
- virtual void **sectionEndedEarly** (SectionEndInfo const &endInfo)=0
- virtual auto **acquireGeneratorTracker** (StringRef generatorName, SourceLineInfo const &lineInfo) -> IGeneratorTracker &=0
- virtual void **pushScopedMessage** (MessageInfo const &message)=0
- virtual void **popScopedMessage** (MessageInfo const &message)=0
- virtual void **emplaceUnscopedMessage** (MessageBuilder const &builder)=0
- virtual void **handleFatalErrorCondition** (StringRef message)=0
- virtual void **handleExpr** (AssertionInfo const &info, ITransientExpression const &expr, AssertionReaction &reaction)=0
- virtual void **handleMessage** (AssertionInfo const &info, ResultWas::OfType resultType, StringRef const &message, AssertionReaction &reaction)=0

- virtual void **handleUnexpectedExceptionNotThrown** (AssertionInfo const &info, AssertionReaction &reaction)=0
- virtual void **handleUnexpectedInflightException** (AssertionInfo const &info, std::string const &message, AssertionReaction &reaction)=0
- virtual void **handleIncomplete** (AssertionInfo const &info)=0
- virtual void **handleNonExpr** (AssertionInfo const &info, ResultWas::OfType resultType, AssertionReaction &reaction)=0
- virtual bool **lastAssertionPassed** ()=0
- virtual void **assertionPassed** ()=0
- virtual std::string **getCurrentTestName** () const =0
- virtual const AssertionResult ∗ **getLastResult** () const =0
- virtual void **exceptionEarlyReported** ()=0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.46 Catch::IRunner Struct Reference

Public Member Functions

- virtual bool **aborting** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.47 Catch::is_callable< T > Struct Template Reference

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.48 Catch::is_callable< Fun(Args...)> Struct Template Reference

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.49 Catch::is_callable_tester Struct Reference

Static Public Member Functions

- template<typename Fun , typename... Args>
  static true_given< decltype(std::declval< Fun >()(std::declval< Args >()...))> **test** (int)
- template<typename... >
  static std::false_type **test** (...)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.50 Catch::is_range< T > Struct Template Reference

Inheritance diagram for Catch::is_range< T >:

struct_catch_1_1is__range-eps-converted-to.pdf

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.51 Catch::detail::is_range_impl< T, typename > Struct Template Reference

Inheritance diagram for Catch::detail::is_range_impl< T, typename >:

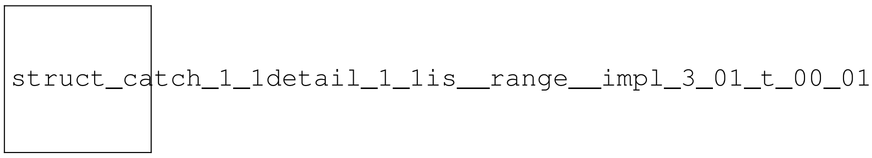
struct_catch_1_1detail_1_1is__range__impl-eps-converted-t

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.52 Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type > Struct Template Reference

Inheritance diagram for Catch::detail::is_range_impl< T, typename void_type< decltype(begin(std::declval< T >()))>::type >:



The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.53 Catch::Detail::IsStreamInsertable< T > Class Template Reference

Static Public Attributes

- static const bool **value** = decltype(test<std::ostream, const T&>(0))::value

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.54 Catch::IStream Struct Reference

Public Member Functions

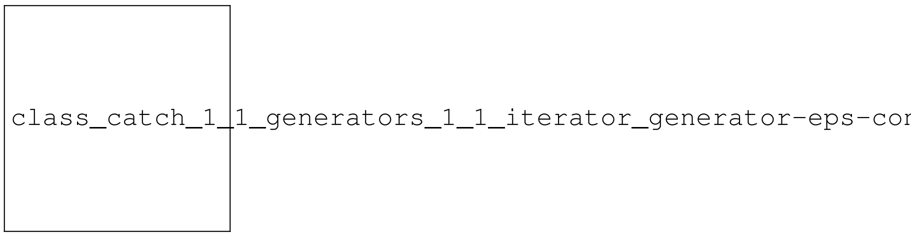
- virtual std::ostream & **stream** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.55 Catch::Generators::IteratorGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::IteratorGenerator< T >:



Public Member Functions

- template<typename InputIterator , typename InputSentinel >
  IteratorGenerator (InputIterator first, InputSentinel last)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.55.1 Member Function Documentation

4.55.1.1 get()

```
template<typename T >
T const & Catch::Generators::IteratorGenerator< T >::get ( ) const  [inline], [override],
[virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.55.1.2 next()

```
template<typename T >
bool Catch::Generators::IteratorGenerator< T >::next ( )  [inline], [override], [virtual]
```

Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.56 Catch::ITestCaseRegistry Struct Reference

Public Member Functions

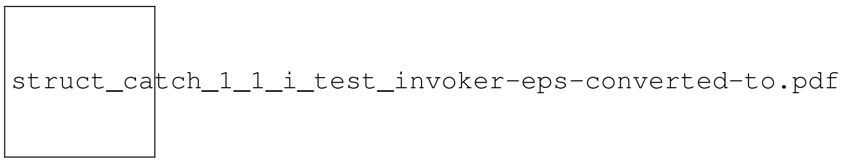
- virtual std::vector< TestCase > const & **getAllTests** () const =0
- virtual std::vector< TestCase > const & **getAllTestsSorted** (IConfig const &config) const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.57 Catch::ITestInvoker Struct Reference

Inheritance diagram for Catch::ITestInvoker:



Public Member Functions

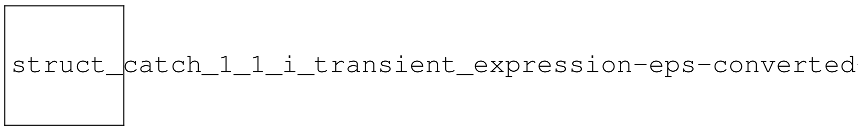
- virtual void **invoke** () const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.58 Catch::ITransientExpression Struct Reference

Inheritance diagram for Catch::ITransientExpression:



Public Member Functions

- auto **isBinaryExpression** () const -> bool
- auto **getResult** () const -> bool
- virtual void **streamReconstructedExpression** (std::ostream &os) const =0
- **ITransientExpression** (bool isBinaryExpression, bool result)

Public Attributes

- bool **m_isBinaryExpression**
- bool **m_result**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.59 Catch::LazyExpression Class Reference

Public Member Functions

- **LazyExpression** (bool isNegated)
- **LazyExpression** (LazyExpression const &other)
- LazyExpression & **operator=** (LazyExpression const &)=delete
- **operator bool** () const

Friends

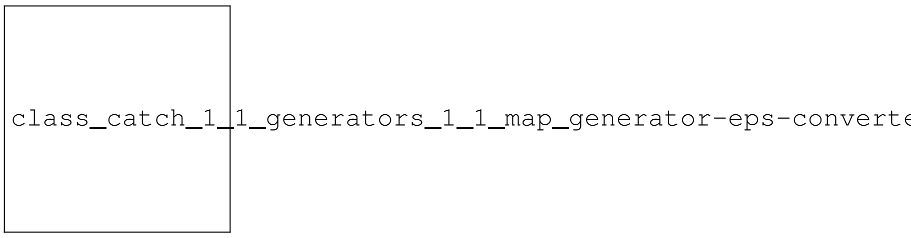
- class **AssertionHandler**
- struct **AssertionStats**
- class **RunContext**
- auto **operator**<< (std::ostream &os, LazyExpression const &lazyExpr) -> std::ostream &

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.60 Catch::Generators::MapGenerator< T, U, Func > Class Template Reference

Inheritance diagram for Catch::Generators::MapGenerator< T, U, Func >:



Public Member Functions

- template<typename F2 = Func>
  MapGenerator (F2 &&function, GeneratorWrapper< U > &&generator)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.60.1 Member Function Documentation

4.60.1.1 get()

```
template<typename T , typename U , typename Func >
T const & Catch::Generators::MapGenerator< T, U, Func >::get ( ) const  [inline], [override],
[virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.60.1.2 next()

```
template<typename T , typename U , typename Func >
bool Catch::Generators::MapGenerator< T, U, Func >::next ( )  [inline], [override], [virtual]
```

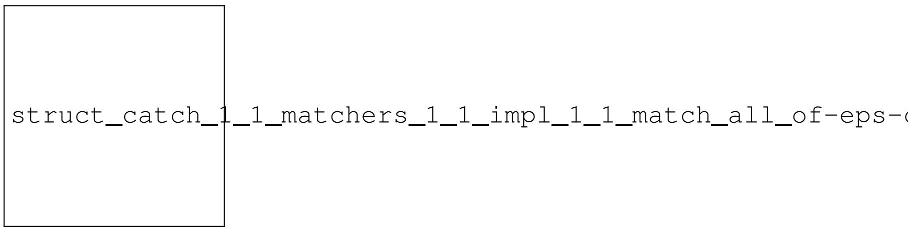
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.61 Catch::Matchers::Impl::MatchAllOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchAllOf< ArgT >:



Public Member Functions

- bool **match** (ArgT const &arg) const override
- std::string describe () const override
- MatchAllOf< ArgT > **operator&&** (MatcherBase< ArgT > const &other)

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< ArgT >

- MatchAllOf< ArgT > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< ArgT > **operator||** (MatcherBase const &other) const
- MatchNotOf< ArgT > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< ObjectT >

- virtual bool **match** (ObjectT const &arg) const =0

Public Attributes

- std::vector< MatcherBase< ArgT > const ∗ > **m_matchers**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.61.1 Member Function Documentation

4.61.1.1 describe()

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchAllOf< ArgT >::describe ( ) const  [inline], [override],
[virtual]
```

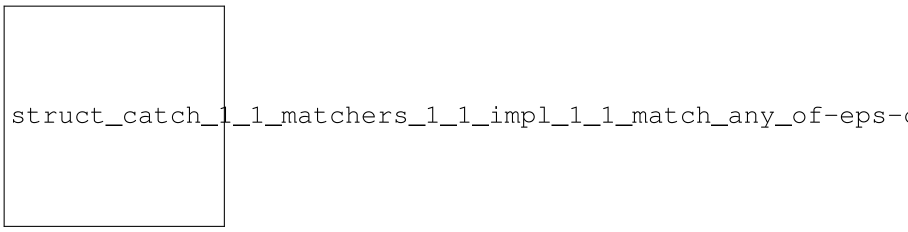
Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.62 Catch::Matchers::Impl::MatchAnyOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchAnyOf< ArgT >:



Public Member Functions

- bool **match** (ArgT const &arg) const override
- std::string describe () const override
- MatchAnyOf< ArgT > **operator**|| (MatcherBase< ArgT > const &other)

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< ArgT >

- MatchAllOf< ArgT > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< ArgT > **operator**|| (MatcherBase const &other) const
- MatchNotOf< ArgT > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< ObjectT >

- virtual bool **match** (ObjectT const &arg) const =0

Public Attributes

- std::vector< MatcherBase< ArgT > const ∗ > **m_matchers**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.62.1 Member Function Documentation

4.62.1.1 describe()

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchAnyOf< ArgT >::describe ( ) const  [inline], [override],
[virtual]
```

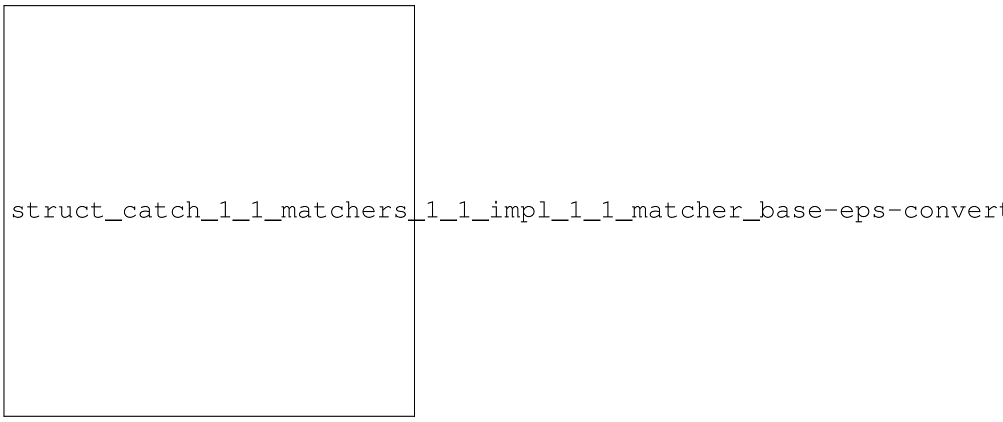
Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.63 Catch::Matchers::Impl::MatcherBase< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherBase< T >:


struct_catch_1_1_matchers_1_1_impl_1_1_matcher_base-eps-convert

Public Member Functions

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Protected Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- virtual std::string **describe** () const =0

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

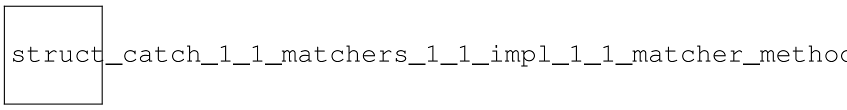
- std::string **m_cachedToString**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.64 Catch::Matchers::Impl::MatcherMethod< ObjectT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherMethod< ObjectT >:



Public Member Functions

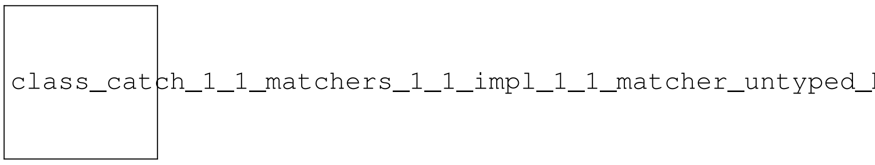
- virtual bool **match** (ObjectT const &arg) const =0

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.65 Catch::Matchers::Impl::MatcherUntypedBase Class Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherUntypedBase:



Public Member Functions

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Protected Member Functions

- virtual std::string **describe** () const =0

Protected Attributes

- std::string **m_cachedToString**

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.66 Catch::MatchExpr< ArgT, MatcherT > Class Template Reference

Inheritance diagram for Catch::MatchExpr< ArgT, MatcherT >:

class_catch_1_1_match_expr-eps-converted-to.pdf

Public Member Functions

- **MatchExpr** (ArgT const &arg, MatcherT const &matcher, StringRef const &matcherString)
- void streamReconstructedExpression (std::ostream &os) const override

Public Member Functions inherited from Catch::ITransientExpression

- auto **isBinaryExpression** () const -> bool
- auto **getResult** () const -> bool
- **ITransientExpression** (bool isBinaryExpression, bool result)

Additional Inherited Members

Public Attributes inherited from Catch::ITransientExpression

- bool **m_isBinaryExpression**
- bool **m_result**

4.66.1 Member Function Documentation

4.66.1.1 streamReconstructedExpression()

```
template<typename ArgT , typename MatcherT >
void Catch::MatchExpr< ArgT, MatcherT >::streamReconstructedExpression (
            std::ostream & os ) const  [inline], [override], [virtual]
```

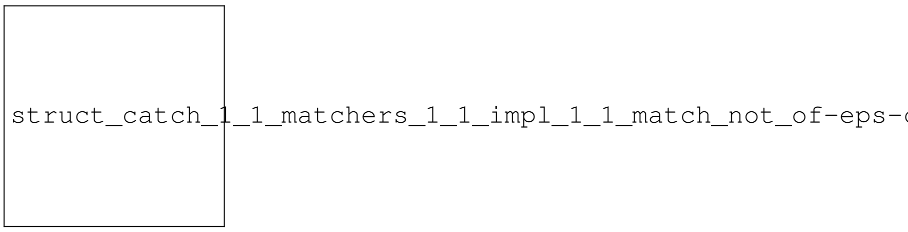
Implements Catch::ITransientExpression.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.67 Catch::Matchers::Impl::MatchNotOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchNotOf< ArgT >:



Public Member Functions

- **MatchNotOf** (MatcherBase< ArgT > const &underlyingMatcher)
- bool **match** (ArgT const &arg) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< ArgT >

- MatchAllOf< ArgT > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< ArgT > **operator||** (MatcherBase const &other) const
- MatchNotOf< ArgT > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< ObjectT >

- virtual bool **match** (ObjectT const &arg) const =0

Public Attributes

- MatcherBase< ArgT > const & **m_underlyingMatcher**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.67.1 Member Function Documentation

4.67.1.1 describe()

```
template<typename ArgT >
std::string Catch::Matchers::Impl::MatchNotOf< ArgT >::describe ( ) const  [inline], [override],
[virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.68 Catch::MessageBuilder Struct Reference

Inheritance diagram for Catch::MessageBuilder:

struct_catch_1_1_message_builder-eps-converted-to.pdf

Public Member Functions

- **MessageBuilder** (StringRef const &macroName, SourceLineInfo const &lineInfo, ResultWas::OfType type)
- template<typename T >
  MessageBuilder & **operator**<< (T const &value)

Public Member Functions inherited from Catch::MessageStream

- template<typename T >
  MessageStream & **operator**<< (T const &value)

Public Attributes

- MessageInfo **m_info**

Public Attributes inherited from Catch::MessageStream

- ReusableStringStream **m_stream**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.69 Catch::MessageInfo Struct Reference

Public Member Functions

- **MessageInfo** (StringRef const &_macroName, SourceLineInfo const &_lineInfo, ResultWas::OfType _type)
- bool **operator==** (MessageInfo const &other) const
- bool **operator<** (MessageInfo const &other) const

Public Attributes

- StringRef **macroName**
- std::string **message**
- SourceLineInfo **lineInfo**
- ResultWas::OfType **type**
- unsigned int **sequence**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.70 Catch::MessageStream Struct Reference

Inheritance diagram for Catch::MessageStream:

struct_catch_1_1_message_stream-eps-converted-to.pdf

Public Member Functions

- template<typename T >
  MessageStream & **operator<<** (T const &value)

Public Attributes

- ReusableStringStream **m_stream**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.71 Mult Class Reference

Inheritance diagram for Mult:

class_mult-eps-converted-to.pdf

Public Member Functions

- Mult (Expr ∗lhs, Expr ∗rhs)

  Constructor for the Mult class.
- bool equals (Expr ∗e)

  Checks if this Mult expression is equal to another expression.
- int interp ()

  Evaluates the multiplication expression.
- bool has_variable ()

  Checks if the expression contains any variables.
- Expr ∗ subst (string varName, Expr ∗replacement)

  Substitutes a variable within the expression with another expression.
- virtual void print (ostream &os)

  Prints the Mult expression in a human-readable form.
- void pretty_print_at (ostream &ot, precedence_t prec)

  Pretty prints the Mult expression with appropriate precedence.

Public Member Functions inherited from Expr

- string **to_string** ()
- void **pretty_print** (ostream &ostream)
- string **to_pretty_string** ()

Public Attributes

- Expr ∗ **lhs**
- Expr ∗ **rhs**

4.71.1 Constructor & Destructor Documentation

4.71.1.1 Mult()

```
Mult::Mult (
            Expr ∗ lhs,
            Expr ∗ rhs )
```

Constructor for the Mult class.

Parameters

lhs	The left-hand side expression of the multiplication.
rhs	The right-hand side expression of the multiplication. Initializes a Mult object with two expressions to be multiplied.

4.71.2 Member Function Documentation

4.71.2.1 equals()

```
bool Mult::equals (
             Expr * e )  [virtual]
```

Checks if this Mult expression is equal to another expression.

Parameters

e	The expression to compare with.

Returns

True if both lhs and rhs of Mult are equal to those of e, false otherwise.

Implements Expr.

4.71.2.2 has_variable()

```
bool Mult::has_variable ( )  [virtual]
```

Checks if the expression contains any variables.

Returns

True if either lhs or rhs contains a variable, false otherwise.

Implements Expr.

4.71.2.3 interp()

```
int Mult::interp ( )  [virtual]
```

Evaluates the multiplication expression.

Returns

The product of the interpretations of lhs and rhs.

Implements Expr.

4.71.2.4 pretty_print_at()

```
void Mult::pretty_print_at (
             ostream & ot,
             precedence_t prec )  [virtual]
```

Pretty prints the Mult expression with appropriate precedence.

Parameters

o	The output stream to print to.
prec	The current precedence level.

Reimplemented from Expr.

4.71.2.5 print()

```
void Mult::print (
             ostream & ostream )  [virtual]
```

Prints the Mult expression in a human-readable form.

Parameters

ostream	The output stream to print to.

Implements Expr.

4.71.2.6 subst()

```
Expr * Mult::subst (
             string varName,
             Expr * replacement )  [virtual]
```

Substitutes a variable within the expression with another expression.

Parameters

varName	The name of the variable to be substituted.
replacement	The expression to substitute in place of the variable.

Returns

A new Mult expression with the variable substituted.

Implements Expr.

The documentation for this class was generated from the following files:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h
- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp

4.72 Catch::NameAndTags Struct Reference

Public Member Functions

- **NameAndTags** (StringRef const &name_=StringRef(), StringRef const &tags_=StringRef()) noexcept

Public Attributes

- StringRef **name**
- StringRef **tags**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.73 Catch::NonCopyable Class Reference

Inheritance diagram for Catch::NonCopyable:

class_catch_1_1_non_copyable-eps-converted-to.pdf

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.74 Num Class Reference

Inheritance diagram for Num:

class_num-eps-converted-to.pdf

Public Member Functions

- Num (int val)

  Constructor for Num.
- bool equals (Expr ∗e)

  Implementation of the equals function for Num.
- int interp ()

  the interp() function for Num class.
- bool has_variable ()

  the has_variable() function for Num class.
- Expr ∗ subst (string varName, Expr ∗replacement)

  The subst() function for Num.
- virtual void print (ostream &os)

  the print function for Num.
- string **to_string** ()

Public Member Functions inherited from Expr

- string **to_string** ()
- void **pretty_print** (ostream &ostream)
- virtual void pretty_print_at (ostream &ot, precedence_t prec)
- string **to_pretty_string** ()

Public Attributes

- int **val**

4.74.1 Constructor & Destructor Documentation

4.74.1.1 Num()

```
Num::Num (
            int val )
```

Constructor for Num.

Parameters

val	The integer value of the Num object. Creates a Num object out of val.

4.74.2 Member Function Documentation

4.74.2.1 equals()

```
bool Num::equals (
            Expr * e )  [virtual]
```

Implementation of the equals function for Num.

Parameters

e	the expression you compare.

Returns

false if num is a null pointer, true otherwise. Verifies the current Num object is equal to a different expression.

Implements Expr.

4.74.2.2 has_variable()

```
bool Num::has_variable ( )  [virtual]
```

the has_variable() function for Num class.

Returns

ALWAYS will return false. Verifies that there are no variables.

Implements Expr.

4.74.2.3 interp()

```
int Num::interp ( )  [virtual]
```

the interp() function for Num class.

Returns

the integer val of Num object.

Implements Expr.

4.74.2.4 print()

```
void Num::print (
             ostream & os )  [virtual]
```

the print function for Num.

Parameters

os	The output stream to print to. Prints the value of the Num object as a string to the specified output stream.

Implements Expr.

4.74.2.5 subst()

```
Expr * Num::subst (
             string varName,
             Expr * replacement )  [virtual]
```

The subst() function for Num.

Parameters

varName	the variable that will be replaced.
replacement	The replacement expression.

Returns

The new expression with the variable substituted. Swaps varName with a replacement expression.

Implements Expr.

The documentation for this class was generated from the following files:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h
- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp

4.75 Catch::Option< T > Class Template Reference

Public Member Functions

- **Option** (T const &_value)
- **Option** (Option const &_other)
- Option & **operator=** (Option const &_other)
- Option & **operator=** (T const &_value)
- void **reset** ()
- T & **operator*** ()
- T const & **operator*** () const
- T * **operator->** ()
- const T * **operator->** () const
- T **valueOr** (T const &defaultValue) const
- bool **some** () const
- bool **none** () const
- bool **operator!** () const
- **operator bool** () const

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.76 Catch::pluralise Struct Reference

Public Member Functions

- **pluralise** (std::size_t count, std::string const &label)

Public Attributes

- std::size_t **m_count**
- std::string **m_label**

Friends

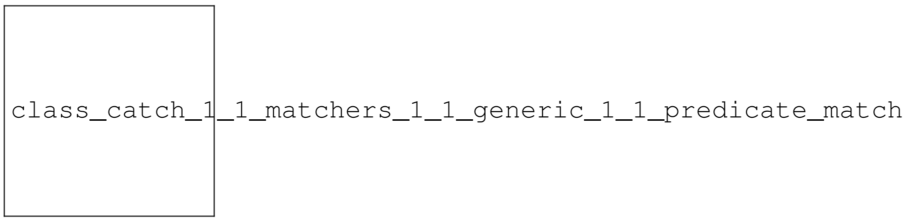
- std::ostream & **operator<<** (std::ostream &os, pluralise const &pluraliser)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.77 Catch::Matchers::Generic::PredicateMatcher< T > Class Template Reference

Inheritance diagram for Catch::Matchers::Generic::PredicateMatcher< T >:



Public Member Functions

- **PredicateMatcher** (std::function< bool(T const &)> const &elem, std::string const &descr)
- bool match (T const &item) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.77.1 Member Function Documentation

4.77.1.1 describe()

```
template<typename T >
std::string Catch::Matchers::Generic::PredicateMatcher< T >::describe ( ) const  [inline],
[override], [virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

4.77.1.2 match()

```
template<typename T >
bool Catch::Matchers::Generic::PredicateMatcher< T >::match (
            T const & item ) const  [inline], [override], [virtual]
```

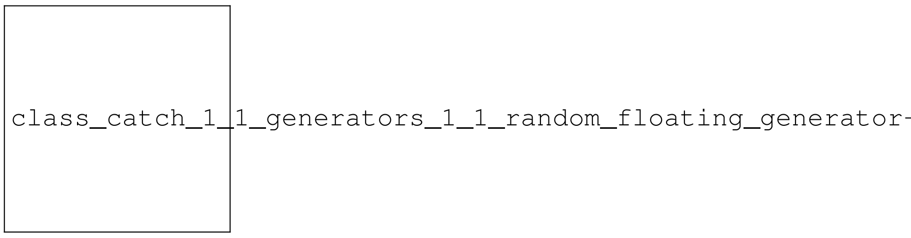
Implements Catch::Matchers::Impl::MatcherMethod< T >.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.78 Catch::Generators::RandomFloatingGenerator< Float > Class Template Reference

Inheritance diagram for Catch::Generators::RandomFloatingGenerator< Float >:



Public Member Functions

- **RandomFloatingGenerator** (Float a, Float b)
- Float const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< Float >

- using **type**

4.78.1 Member Function Documentation

4.78.1.1 get()

```
template<typename Float >
Float const & Catch::Generators::RandomFloatingGenerator< Float >::get ( ) const  [inline],
[override], [virtual]
```

Implements Catch::Generators::IGenerator< Float >.

4.78.1.2 next()

```
template<typename Float >
bool Catch::Generators::RandomFloatingGenerator< Float >::next ( )  [inline], [override],
[virtual]
```

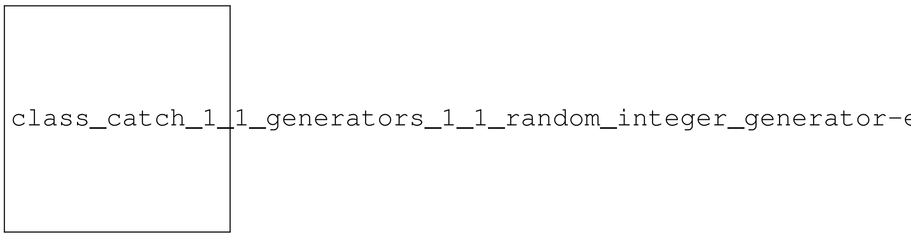
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.79 Catch::Generators::RandomIntegerGenerator< Integer > Class Template Reference

Inheritance diagram for Catch::Generators::RandomIntegerGenerator< Integer >:



Public Member Functions

- **RandomIntegerGenerator** (Integer a, Integer b)
- Integer const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< Integer >

- using **type**

4.79.1 Member Function Documentation

4.79.1.1 get()

```
template<typename Integer >
Integer const & Catch::Generators::RandomIntegerGenerator< Integer >::get ( ) const  [inline],
[override], [virtual]
```

Implements Catch::Generators::IGenerator< Integer >.

4.79.1.2 next()

```
template<typename Integer >
bool Catch::Generators::RandomIntegerGenerator< Integer >::next ( )  [inline], [override],
[virtual]
```

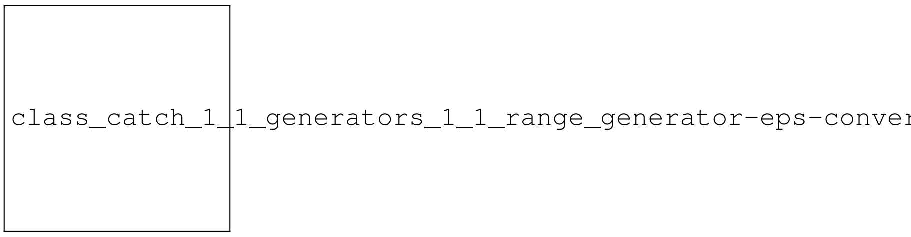
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.80 Catch::Generators::RangeGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::RangeGenerator< T >:



Public Member Functions

- **RangeGenerator** (T const &start, T const &end, T const &step)
- **RangeGenerator** (T const &start, T const &end)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.80.1 Member Function Documentation

4.80.1.1 get()

```
template<typename T >
T const & Catch::Generators::RangeGenerator< T >::get ( ) const  [inline], [override], [virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.80.1.2 next()

```
template<typename T >
bool Catch::Generators::RangeGenerator< T >::next ( )  [inline], [override], [virtual]
```

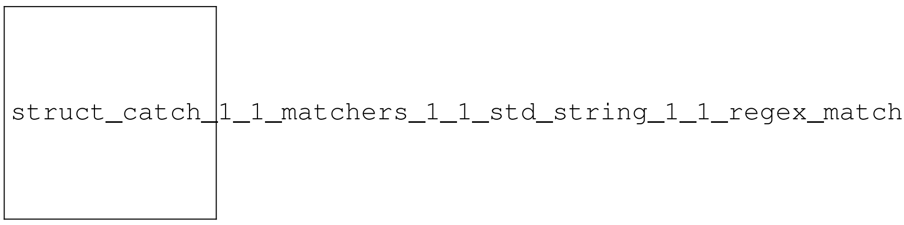
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.81 Catch::Matchers::StdString::RegexMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::RegexMatcher:



Public Member Functions

- **RegexMatcher** (std::string regex, CaseSensitive::Choice caseSensitivity)
- bool **match** (std::string const &matchee) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.81.1 Member Function Documentation

4.81.1.1 describe()

```
std::string Catch::Matchers::StdString::RegexMatcher::describe ( ) const  [override], [virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.82 Catch::RegistrarForTagAliases Struct Reference

Public Member Functions

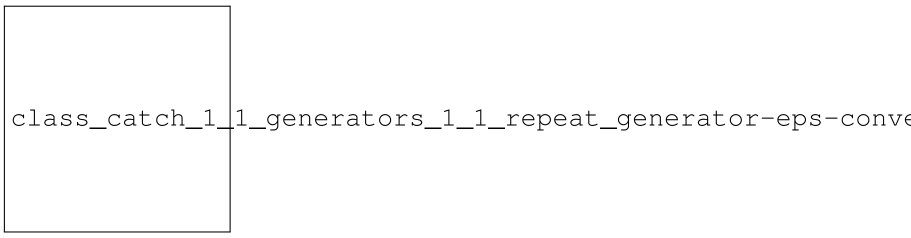
- **RegistrarForTagAliases** (char const ∗alias, char const ∗tag, SourceLineInfo const &lineInfo)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.83 Catch::Generators::RepeatGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::RepeatGenerator< T >:



Public Member Functions

- **RepeatGenerator** (size_t repeats, GeneratorWrapper< T > &&generator)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.83.1 Member Function Documentation

4.83.1.1 get()

```
template<typename T >
T const & Catch::Generators::RepeatGenerator< T >::get ( ) const  [inline], [override], [virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.83.1.2 next()

```
template<typename T >
bool Catch::Generators::RepeatGenerator< T >::next ( )  [inline], [override], [virtual]
```

Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.84 Catch::ResultDisposition Struct Reference

Public Types

- enum **Flags** { **Normal** = 0x01 , **ContinueOnFailure** = 0x02 , **FalseTest** = 0x04 , **SuppressFail** = 0x08 }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.85 Catch::ResultWas Struct Reference

Public Types

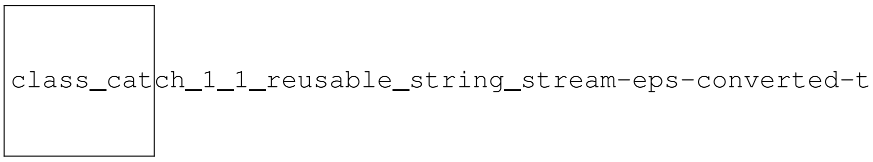
- enum **OfType** {
  Unknown = -1 , **Ok** = 0 , **Info** = 1 , **Warning** = 2 ,
  FailureBit = 0x10 , **ExpressionFailed** = FailureBit | 1 , **ExplicitFailure** = FailureBit | 2 , **Exception** = 0x100 | FailureBit ,
  ThrewException = Exception | 1 , **DidntThrowException** = Exception | 2 , **FatalErrorCondition** = 0x200 | FailureBit }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.86 Catch::ReusableStringStream Class Reference

Inheritance diagram for Catch::ReusableStringStream:



Public Member Functions

- auto **str** () const -> std::string
- template<typename T >
  auto **operator**<< (T const &value) -> ReusableStringStream &
- auto **get** () -> std::ostream &

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.87 Catch::RunTests Struct Reference

Public Types

- enum **InWhatOrder** { **InDeclarationOrder** , **InLexicographicalOrder** , **InRandomOrder** }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.88 Catch::ScopedMessage Class Reference

Public Member Functions

- **ScopedMessage** (MessageBuilder const &builder)
- **ScopedMessage** (ScopedMessage &duplicate)=delete
- **ScopedMessage** (ScopedMessage &&old)

Public Attributes

- MessageInfo **m_info**
- bool **m_moved**

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.89 Catch::Section Class Reference

Inheritance diagram for Catch::Section:

Public Member Functions

- **Section** (SectionInfo const &info)
- **operator bool** () const

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.90 Catch::SectionEndInfo Struct Reference

Public Attributes

- SectionInfo **sectionInfo**
- Counts **prevAssertions**
- double **durationInSeconds**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.91 Catch::SectionInfo Struct Reference

Public Member Functions

- **SectionInfo** (SourceLineInfo const &_lineInfo, std::string const &_name)
- **SectionInfo** (SourceLineInfo const &_lineInfo, std::string const &_name, std::string const &)

Public Attributes

- std::string **name**
- std::string **description**
- SourceLineInfo **lineInfo**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.92 Catch::ShowDurations Struct Reference

Public Types

- enum **OrNot** { **DefaultForReporter** , **Always** , **Never** }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.93 Catch::SimplePcg32 Class Reference

Public Types

- using **result_type** = std::uint32_t

Public Member Functions

- **SimplePcg32** (result_type seed_)
- void **seed** (result_type seed_)
- void **discard** (uint64_t skip)
- result_type **operator()** ()

Static Public Member Functions

- static constexpr result_type **min** ()
- static constexpr result_type **max** ()

Friends

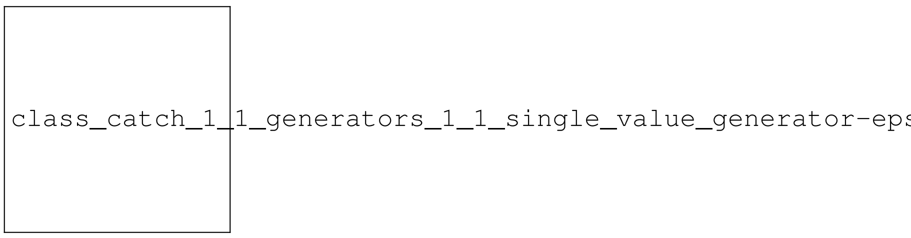
- bool **operator==** (SimplePcg32 const &lhs, SimplePcg32 const &rhs)
- bool **operator!=** (SimplePcg32 const &lhs, SimplePcg32 const &rhs)

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.94 Catch::Generators::SingleValueGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::SingleValueGenerator< T >:



Public Member Functions

- **SingleValueGenerator** (T &&value)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.94.1 Member Function Documentation

4.94.1.1 get()

```
template<typename T >
T const & Catch::Generators::SingleValueGenerator< T >::get ( ) const  [inline], [override],
[virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.94.1.2 next()

```
template<typename T >
bool Catch::Generators::SingleValueGenerator< T >::next ( )  [inline], [override], [virtual]
```

Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.95 Catch::SourceLineInfo Struct Reference

Public Member Functions

- **SourceLineInfo** (char const ∗_file, std::size_t _line) noexcept
- **SourceLineInfo** (SourceLineInfo const &other)=default
- SourceLineInfo & **operator=** (SourceLineInfo const &)=default
- **SourceLineInfo** (SourceLineInfo &&) noexcept=default
- SourceLineInfo & **operator=** (SourceLineInfo &&) noexcept=default
- bool **empty** () const noexcept
- bool **operator==** (SourceLineInfo const &other) const noexcept
- bool **operator<** (SourceLineInfo const &other) const noexcept

Public Attributes

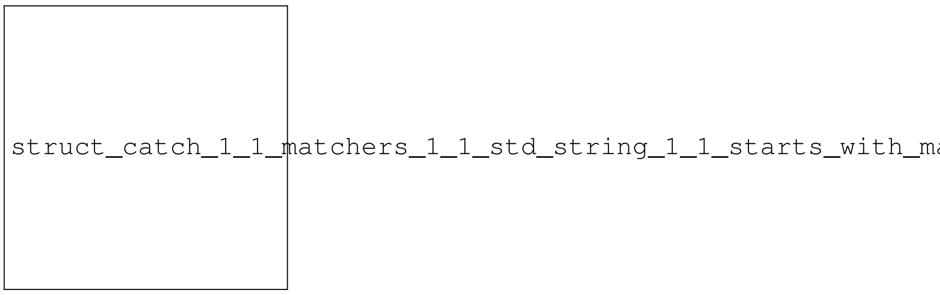
- char const ∗ **file**
- std::size_t **line**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.96 Catch::Matchers::StdString::StartsWithMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::StartsWithMatcher:



Public Member Functions

- **StartsWithMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Public Member Functions inherited from Catch::Matchers::StdString::StringMatcherBase

- **StringMatcherBase** (std::string const &operation, CasedString const &comparator)
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator**|| (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Public Attributes inherited from Catch::Matchers::StdString::StringMatcherBase

- CasedString **m_comparator**
- std::string **m_operation**

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.97 Catch::StreamEndStop Struct Reference

Public Member Functions

- std::string **operator+** () const

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.98 Catch::StringMaker< T, typename > Struct Template Reference

Static Public Member Functions

- template<typename Fake = T>
  static std::enable_if<::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type **convert** (const Fake &value)
- template<typename Fake = T>
  static std::enable_if<!::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type **convert** (const Fake &value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.99 Catch::StringMaker< bool > Struct Reference

Static Public Member Functions

- static std::string **convert** (bool b)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.100 Catch::StringMaker< Catch::Detail::Approx > Struct Reference

Static Public Member Functions

- static std::string **convert** (Catch::Detail::Approx const &value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.101 Catch::StringMaker< char * > Struct Reference

Static Public Member Functions

- static std::string **convert** (char *str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.102 Catch::StringMaker< char > Struct Reference

Static Public Member Functions

- static std::string **convert** (char c)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.103 Catch::StringMaker< char const * > Struct Reference

Static Public Member Functions

- static std::string **convert** (char const *str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.104 Catch::StringMaker< char[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (char const *str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.105 Catch::StringMaker< double > Struct Reference

Static Public Member Functions

- static std::string **convert** (double value)

Static Public Attributes

- static int **precision**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.106 Catch::StringMaker< float > Struct Reference

Static Public Member Functions

- static std::string **convert** (float value)

Static Public Attributes

- static int **precision**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.107 Catch::StringMaker< int > Struct Reference

Static Public Member Functions

- static std::string **convert** (int value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.108 Catch::StringMaker< long > Struct Reference

Static Public Member Functions

- static std::string **convert** (long value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.109 Catch::StringMaker< long long > Struct Reference

Static Public Member Functions

- static std::string **convert** (long long value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.110 Catch::StringMaker< R C::* > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (R C::*p)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.111 Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (R const &range)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.112 Catch::StringMaker< signed char > Struct Reference

Static Public Member Functions

- static std::string **convert** (signed char c)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.113 Catch::StringMaker< signed char[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (signed char const *str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.114 Catch::StringMaker< std::nullptr_t > Struct Reference

Static Public Member Functions

- static std::string **convert** (std::nullptr_t)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.115 Catch::StringMaker< std::string > Struct Reference

Static Public Member Functions

- static std::string **convert** (const std::string &str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.116 Catch::StringMaker< std::wstring > Struct Reference

Static Public Member Functions

- static std::string **convert** (const std::wstring &wstr)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.117 Catch::StringMaker< T * > Struct Template Reference

Static Public Member Functions

- template<typename U >
  static std::string **convert** (U *p)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.118 Catch::StringMaker< T[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (T const(&arr)[SZ])

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.119 Catch::StringMaker< unsigned char > Struct Reference

Static Public Member Functions

- static std::string **convert** (unsigned char c)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.120 Catch::StringMaker< unsigned char[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (unsigned char const ∗str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.121 Catch::StringMaker< unsigned int > Struct Reference

Static Public Member Functions

- static std::string **convert** (unsigned int value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.122 Catch::StringMaker< unsigned long > Struct Reference

Static Public Member Functions

- static std::string **convert** (unsigned long value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.123 Catch::StringMaker< unsigned long long > Struct Reference

Static Public Member Functions

- static std::string **convert** (unsigned long long value)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.124 Catch::StringMaker< wchar_t * > Struct Reference

Static Public Member Functions

- static std::string **convert** (wchar_t *str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.125 Catch::StringMaker< wchar_t const * > Struct Reference

Static Public Member Functions

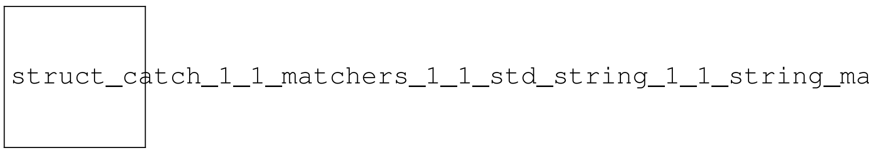
- static std::string **convert** (wchar_t const *str)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.126 Catch::Matchers::StdString::StringMatcherBase Struct Reference

Inheritance diagram for Catch::Matchers::StdString::StringMatcherBase:



Public Member Functions

- **StringMatcherBase** (std::string const &operation, CasedString const &comparator)
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Public Attributes

- CasedString **m_comparator**
- std::string **m_operation**

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.126.1 Member Function Documentation

4.126.1.1 describe()

```
std::string Catch::Matchers::StdString::StringMatcherBase::describe ( ) const  [override],
[virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.127 Catch::StringRef Class Reference

```
#include <catch.h>
```

Public Types

- using **size_type** = std::size_t
- using **const_iterator** = const char∗

Public Member Functions

- **StringRef** (char const ∗rawChars) noexcept
- constexpr **StringRef** (char const ∗rawChars, size_type size) noexcept
- **StringRef** (std::string const &stdString) noexcept
- **operator std::string** () const
- auto **operator==** (StringRef const &other) const noexcept -> bool
- auto **operator!=** (StringRef const &other) const noexcept -> bool
- auto **operator[ ]** (size_type index) const noexcept -> char
- constexpr auto **empty** () const noexcept -> bool
- constexpr auto **size** () const noexcept -> size_type
- auto **c_str** () const -> char const ∗
- auto **substr** (size_type start, size_type length) const noexcept -> StringRef
- auto **data** () const noexcept -> char const ∗
- constexpr auto **isNullTerminated** () const noexcept -> bool
- constexpr const_iterator **begin** () const
- constexpr const_iterator **end** () const

4.127.1 Detailed Description

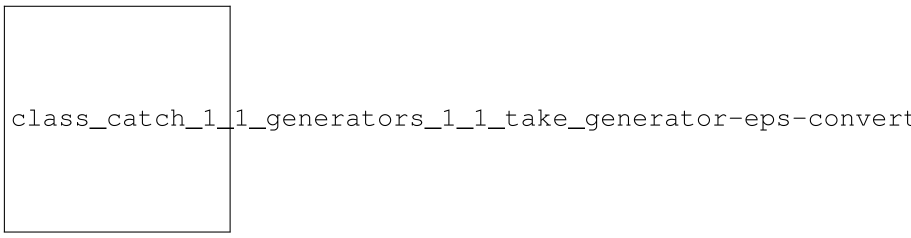
A non-owning string class (similar to the forthcoming std::string_view) Note that, because a StringRef may be a substring of another string, it may not be null terminated.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.128 Catch::Generators::TakeGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::TakeGenerator< T >:



Public Member Functions

- **TakeGenerator** (size_t target, GeneratorWrapper< T > &&generator)
- T const & get () const override
- bool next () override

Additional Inherited Members

Public Types inherited from Catch::Generators::IGenerator< T >

- using **type** = T

4.128.1 Member Function Documentation

4.128.1.1 get()

```
template<typename T >
T const & Catch::Generators::TakeGenerator< T >::get ( ) const  [inline], [override], [virtual]
```

Implements Catch::Generators::IGenerator< T >.

4.128.1.2 next()

```
template<typename T >
bool Catch::Generators::TakeGenerator< T >::next ( )  [inline], [override], [virtual]
```

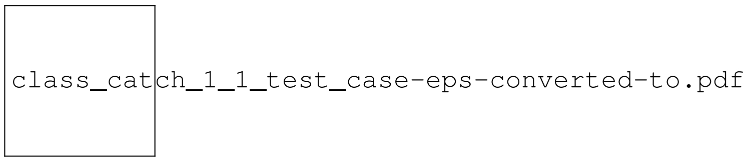
Implements Catch::Generators::GeneratorUntypedBase.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.129 Catch::TestCase Class Reference

Inheritance diagram for Catch::TestCase:



Public Member Functions

- **TestCase** (ITestInvoker ∗testCase, TestCaseInfo &&info)
- TestCase **withName** (std::string const &_newName) const
- void **invoke** () const
- TestCaseInfo const & **getTestCaseInfo** () const
- bool **operator==** (TestCase const &other) const
- bool **operator<** (TestCase const &other) const

Public Member Functions inherited from Catch::TestCaseInfo

- **TestCaseInfo** (std::string const &_name, std::string const &_className, std::string const &_description, std::vector< std::string > const &_tags, SourceLineInfo const &_lineInfo)
- bool **isHidden** () const
- bool **throws** () const
- bool **okToFail** () const
- bool **expectedToFail** () const
- std::string **tagsAsString** () const

Additional Inherited Members

Public Types inherited from Catch::TestCaseInfo

- enum **SpecialProperties** {
  None = 0 , **IsHidden** = 1 << 1 , **ShouldFail** = 1 << 2 , **MayFail** = 1 << 3 ,
  Throws = 1 << 4 , **NonPortable** = 1 << 5 , **Benchmark** = 1 << 6 }

Public Attributes inherited from Catch::TestCaseInfo

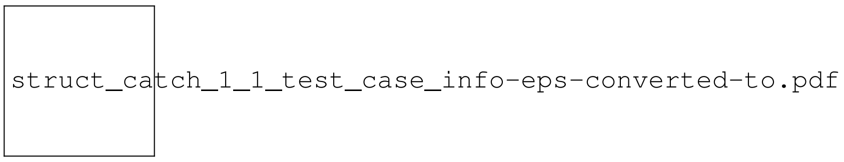
- std::string **name**
- std::string **className**
- std::string **description**
- std::vector< std::string > **tags**
- std::vector< std::string > **lcaseTags**
- SourceLineInfo **lineInfo**
- SpecialProperties **properties**

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.130 Catch::TestCaseInfo Struct Reference

Inheritance diagram for Catch::TestCaseInfo:



Public Types

- enum **SpecialProperties** {
  None = 0 , **IsHidden** = 1 << 1 , **ShouldFail** = 1 << 2 , **MayFail** = 1 << 3 ,
  Throws = 1 << 4 , **NonPortable** = 1 << 5 , **Benchmark** = 1 << 6 }

Public Member Functions

- **TestCaseInfo** (std::string const &_name, std::string const &_className, std::string const &_description, std::vector< std::string > const &_tags, SourceLineInfo const &_lineInfo)
- bool **isHidden** () const
- bool **throws** () const
- bool **okToFail** () const
- bool **expectedToFail** () const
- std::string **tagsAsString** () const

Public Attributes

- std::string **name**
- std::string **className**
- std::string **description**
- std::vector< std::string > **tags**
- std::vector< std::string > **lcaseTags**
- SourceLineInfo **lineInfo**
- SpecialProperties **properties**

Friends

- void **setTags** (TestCaseInfo &testCaseInfo, std::vector< std::string > tags)

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

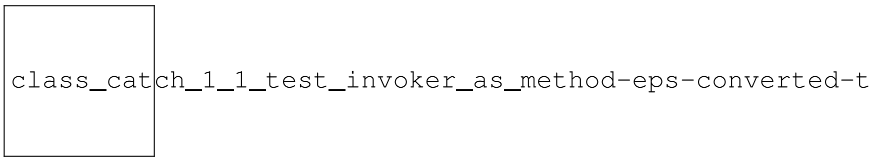
4.131 Catch::TestFailureException Struct Reference

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.132 Catch::TestInvokerAsMethod< C > Class Template Reference

Inheritance diagram for Catch::TestInvokerAsMethod< C >:



Public Member Functions

- **TestInvokerAsMethod** (void(C::∗testAsMethod)()) noexcept
- void invoke () const override

4.132.1 Member Function Documentation

4.132.1.1 invoke()

```
template<typename C >
void Catch::TestInvokerAsMethod< C >::invoke ( ) const  [inline], [override], [virtual]
```

Implements Catch::ITestInvoker.

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.133 Catch::Timer Class Reference

Public Member Functions

- void **start** ()
- auto **getElapsedNanoseconds** () const -> uint64_t
- auto **getElapsedMicroseconds** () const -> uint64_t
- auto **getElapsedMilliseconds** () const -> unsigned int
- auto **getElapsedSeconds** () const -> double

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.134 Catch::Totals Struct Reference

Public Member Functions

- Totals **operator-** (Totals const &other) const
- Totals & **operator+=** (Totals const &other)
- Totals **delta** (Totals const &prevTotals) const

Public Attributes

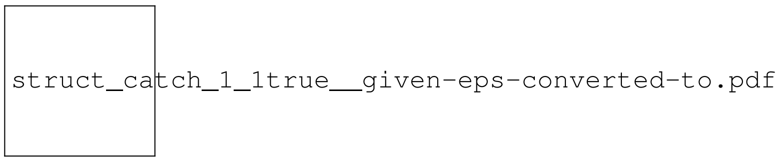
- int **error** = 0
- Counts **assertions**
- Counts **testCases**

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.135 Catch::true_given< typename > Struct Template Reference

Inheritance diagram for Catch::true_given< typename >:

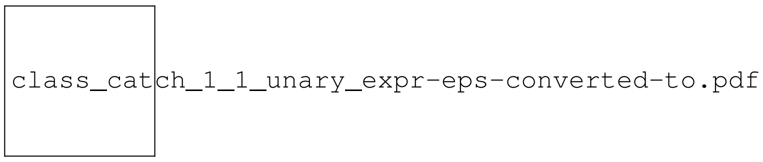


The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.136 Catch::UnaryExpr< LhsT > Class Template Reference

Inheritance diagram for Catch::UnaryExpr< LhsT >:



Public Member Functions

- **UnaryExpr** (LhsT lhs)

Public Member Functions inherited from Catch::ITransientExpression

- auto **isBinaryExpression** () const -> bool
- auto **getResult** () const -> bool
- **ITransientExpression** (bool isBinaryExpression, bool result)

Additional Inherited Members

Public Attributes inherited from Catch::ITransientExpression

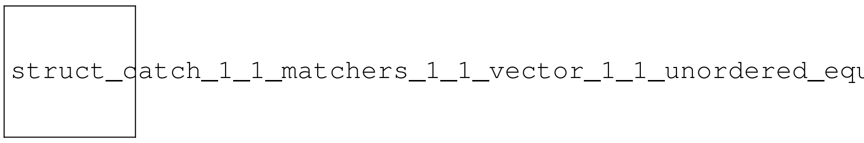
- bool **m_isBinaryExpression**
- bool **m_result**

The documentation for this class was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.137 Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >:



Public Member Functions

- **UnorderedEqualsMatcher** (std::vector< T, AllocComp > const &target)
- bool **match** (std::vector< T, AllocMatch > const &vec) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.137.1 Member Function Documentation

4.137.1.1 describe()

```
template<typename T , typename AllocComp , typename AllocMatch >
std::string Catch::Matchers::Vector::UnorderedEqualsMatcher< T, AllocComp, AllocMatch >↩
::describe ( ) const  [inline], [override], [virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.138 Catch::UseColour Struct Reference

Public Types

- enum **YesOrNo** { **Auto** , **Yes** , **No** }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.139 Var Class Reference

Inheritance diagram for Var:

Public Member Functions

- Var (string name)

  Constructor for Var.
- bool equals (Expr ∗e)

  Implementation of the equals function for Var.
- int interp ()

  the interp() function for Var class.
- bool has_variable ()

  the has_variable() function for Var class.
- Expr ∗ subst (string varName, Expr ∗replacement)

  The subst() function for Var.
- virtual void print (ostream &os)

  the print function for Var.

Public Member Functions inherited from Expr

- string **to_string** ()
- void **pretty_print** (ostream &ostream)
- virtual void pretty_print_at (ostream &ot, precedence_t prec)
- string **to_pretty_string** ()

Public Attributes

- string **name**

4.139.1 Constructor & Destructor Documentation

4.139.1.1 Var()

```
Var::Var (
            string name )
```

Constructor for Var.

Parameters

name	The integer value of the Num object. Creates a Num object out of val.

4.139.2 Member Function Documentation

4.139.2.1 equals()

```
bool Var::equals (
            Expr ∗ e )  [virtual]
```

Implementation of the equals function for Var.

Parameters

e	the expression you compare.

Returns

true if name is equal, false otherwise. Verifies the current Var object is equal to a different name.

Implements Expr.

4.139.2.2 has_variable()

```
bool Var::has_variable ( )  [virtual]
```

the has_variable() function for Var class.

Returns

ALWAYS will return true. Verifies that a variable is a variable.

Implements Expr.

4.139.2.3 interp()

```
int Var::interp ( )  [virtual]
```

the interp() function for Var class.

Returns

runtime error.

Implements Expr.

4.139.2.4 print()

```
void Var::print (
            ostream & ostream )  [virtual]
```

the print function for Var.

Parameters

ostream	The output stream to print to. Prints the value of the Var object to the specified output stream.

Implements Expr.

4.139.2.5 subst()

```
Expr * Var::subst (
            string varName,
            Expr * replacement )  [virtual]
```

The subst() function for Var.

Parameters

varName	the variable that will be replaced.
replacement	The replacement expression.

Returns

the replacement, or a new Var instance with the same name.

Implements Expr.

The documentation for this class was generated from the following files:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h
- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp

4.140 Catch::detail::void_type< ... > Struct Template Reference

Public Types

- using **type** = void

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.141 Catch::WaitForKeypress Struct Reference

Public Types

- enum **When** { **Never** , **BeforeStart** = 1 , **BeforeExit** = 2 , **BeforeStartAndExit** = BeforeStart | BeforeExit }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.142 Catch::WarnAbout Struct Reference

Public Types

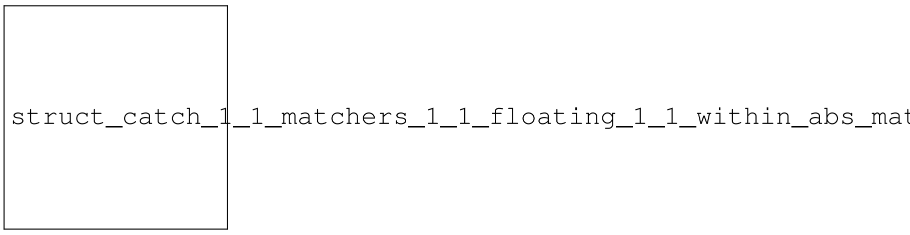
- enum **What** { **Nothing** = 0x00 , **NoAssertions** = 0x01 , **NoTests** = 0x02 }

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.143 Catch::Matchers::Floating::WithinAbsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinAbsMatcher:



Public Member Functions

- **WithinAbsMatcher** (double target, double margin)
- bool **match** (double const &matchee) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.143.1 Member Function Documentation

4.143.1.1 describe()

`std::string Catch::Matchers::Floating::WithinAbsMatcher::describe ( ) const [override], [virtual]`

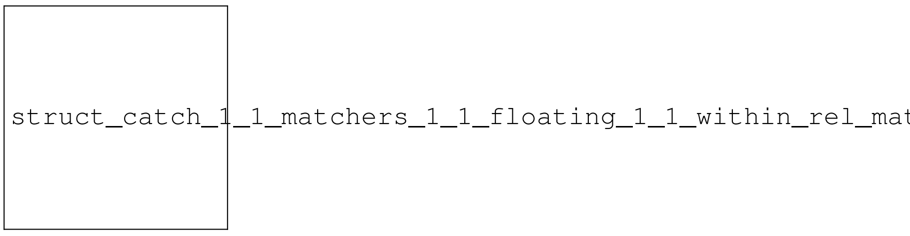
Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.144 Catch::Matchers::Floating::WithinRelMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinRelMatcher:



Public Member Functions

- **WithinRelMatcher** (double target, double epsilon)
- bool **match** (double const &matchee) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator**|| (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.144.1 Member Function Documentation

4.144.1.1 describe()

```
std::string Catch::Matchers::Floating::WithinRelMatcher::describe ( ) const  [override], [virtual]
```

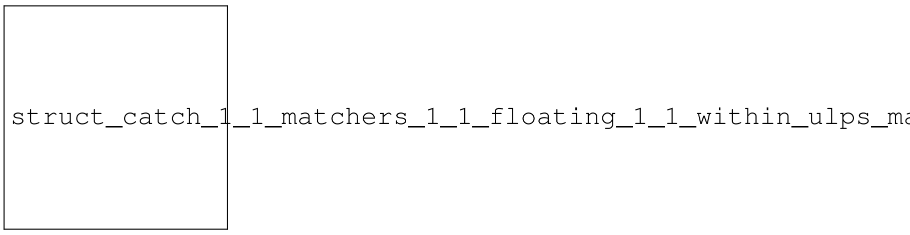
Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

4.145 Catch::Matchers::Floating::WithinUlpsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinUlpsMatcher:



Public Member Functions

- **WithinUlpsMatcher** (double target, uint64_t ulps, FloatingPointKind baseType)
- bool **match** (double const &matchee) const override
- std::string describe () const override

Public Member Functions inherited from Catch::Matchers::Impl::MatcherBase< T >

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherUntypedBase

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Public Member Functions inherited from Catch::Matchers::Impl::MatcherMethod< T >

- virtual bool **match** (T const &arg) const=0

Additional Inherited Members

Protected Attributes inherited from Catch::Matchers::Impl::MatcherUntypedBase

- std::string **m_cachedToString**

4.145.1 Member Function Documentation

4.145.1.1 describe()

```
std::string Catch::Matchers::Floating::WithinUlpsMatcher::describe ( ) const  [override],
[virtual]
```

Implements Catch::Matchers::Impl::MatcherUntypedBase.

The documentation for this struct was generated from the following file:

- /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

Chapter 5

File Documentation

5.1 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/catch.h

```
/*
 *  Catch v2.13.10
 *  Generated: 2022-10-16 11:01:23.452308
 *  ----------------------------------------------------------
 *  This file has been merged from multiple headers. Please don't edit it directly
 *  Copyright (c) 2022 Two Blue Cubes Ltd. All rights reserved.
 *
 *  Distributed under the Boost Software License, Version 1.0. (See accompanying
 *  file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
 */
#ifndef TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
#define TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
// start catch.hpp


#define CATCH_VERSION_MAJOR 2
#define CATCH_VERSION_MINOR 13
#define CATCH_VERSION_PATCH 10

#ifdef __clang__
#    pragma clang system_header
#elif defined __GNUC__
#    pragma GCC system_header
#endif

// start catch_suppress_warnings.h

#ifdef __clang__
#   ifdef __ICC // icpc defines the __clang__ macro
#       pragma warning(push)
#       pragma warning(disable: 161 1682)
#   else // __ICC
#       pragma clang diagnostic push
#       pragma clang diagnostic ignored "-Wpadded"
#       pragma clang diagnostic ignored "-Wswitch-enum"
#       pragma clang diagnostic ignored "-Wcovered-switch-default"
#    endif
#elif defined __GNUC__
     // Because REQUIREs trigger GCC's -Wparentheses, and because still
     // supported version of g++ have only buggy support for _Pragmas,
     // Wparentheses have to be suppressed globally.
#    pragma GCC diagnostic ignored "-Wparentheses" // See #674 for details

#    pragma GCC diagnostic push
#    pragma GCC diagnostic ignored "-Wunused-variable"
#    pragma GCC diagnostic ignored "-Wpadded"
#endif
// end catch_suppress_warnings.h
#if defined(CATCH_CONFIG_MAIN) || defined(CATCH_CONFIG_RUNNER)
#  define CATCH_IMPL
#  define CATCH_CONFIG_ALL_PARTS
#endif

// In the impl file, we want to have access to all parts of the headers
// Can also be used to sanely support PCHs
#if defined(CATCH_CONFIG_ALL_PARTS)
#  define CATCH_CONFIG_EXTERNAL_INTERFACES
#  if defined(CATCH_CONFIG_DISABLE_MATCHERS)
#    undef CATCH_CONFIG_DISABLE_MATCHERS
#  endif
#  if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
#    define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
#  endif
#endif

#if !defined(CATCH_CONFIG_IMPL_ONLY)
// start catch_platform.h

// See e.g.:
// https://opensource.apple.com/source/CarbonHeaders/CarbonHeaders-18.1/TargetConditionals.h.auto.html
#ifdef __APPLE__
#  include <TargetConditionals.h>
#  if (defined(TARGET_OS_OSX) && TARGET_OS_OSX == 1) || \
      (defined(TARGET_OS_MAC) && TARGET_OS_MAC == 1)
#    define CATCH_PLATFORM_MAC
#  elif (defined(TARGET_OS_IPHONE) && TARGET_OS_IPHONE == 1)
#    define CATCH_PLATFORM_IPHONE
#  endif

#elif defined(linux) || defined(__linux) || defined(__linux__)
#  define CATCH_PLATFORM_LINUX

#elif defined(WIN32) || defined(__WIN32__) || defined(_WIN32) || defined(_MSC_VER) ||
defined(__MINGW32__)
#  define CATCH_PLATFORM_WINDOWS
#endif

// end catch_platform.h

#ifdef CATCH_IMPL
#  ifndef CLARA_CONFIG_MAIN
#    define CLARA_CONFIG_MAIN_NOT_DEFINED
#    define CLARA_CONFIG_MAIN
#  endif
#endif

// start catch_user_interfaces.h

namespace Catch {
    unsigned int rngSeed();
}

// end catch_user_interfaces.h
// start catch_tag_alias_autoregistrar.h

// start catch_common.h

// start catch_compiler_capabilities.h

// Detect a number of compiler features - by compiler
// The following features are defined:
//
// CATCH_CONFIG_COUNTER : is the __COUNTER__ macro supported?
// CATCH_CONFIG_WINDOWS_SEH : is Windows SEH supported?
// CATCH_CONFIG_POSIX_SIGNALS : are POSIX signals supported?
// CATCH_CONFIG_DISABLE_EXCEPTIONS : Are exceptions enabled?
// ****************
// Note to maintainers: if new toggles are added please document them
// in configuration.md, too
// ****************

// In general each macro has a _NO_<feature name> form
// (e.g. CATCH_CONFIG_NO_POSIX_SIGNALS) which disables the feature.
// Many features, at point of detection, define an _INTERNAL_ macro, so they
// can be combined, en-mass, with the _NO_ forms later.

#ifdef __cplusplus

#  if (__cplusplus >= 201402L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201402L)
#    define CATCH_CPP14_OR_GREATER
#  endif

#  if (__cplusplus >= 201703L) || (defined(_MSVC_LANG) && _MSVC_LANG >= 201703L)
#    define CATCH_CPP17_OR_GREATER
#  endif

#endif

// Only GCC compiler should be used in this block, so other compilers trying to
// mask themselves as GCC should be ignored.
#if defined(__GNUC__) && !defined(__clang__) && !defined(__ICC) && !defined(__CUDACC__) &&
!defined(__LCC__)
#    define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "GCC diagnostic push" )
#    define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION  _Pragma( "GCC diagnostic pop" )

#    define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__)

#endif

#if defined(__clang__)

#    define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION _Pragma( "clang diagnostic push" )
#    define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION  _Pragma( "clang diagnostic pop" )

// As of this writing, IBM XL's implementation of __builtin_constant_p has a bug
// which results in calls to destructors being emitted for each temporary,
// without a matching initialization. In practice, this can result in something
// like `std::string::~string` being called on an uninitialized value.
//
// For example, this code will likely segfault under IBM XL:
// ```
// REQUIRE(std::string("12") + "34" == "1234")
// ```
//
// Therefore, `CATCH_INTERNAL_IGNORE_BUT_WARN` is not implemented.
#  if !defined(__ibmxl__) && !defined(__CUDACC__)
#    define CATCH_INTERNAL_IGNORE_BUT_WARN(...) (void)__builtin_constant_p(__VA_ARGS__) /*
NOLINT(cppcoreguidelines-pro-type-vararg, hicpp-vararg) */
#  endif

#    define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
         _Pragma( "clang diagnostic ignored \"-Wexit-time-destructors\"" ) \
         _Pragma( "clang diagnostic ignored \"-Wglobal-constructors\"")

#    define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
         _Pragma( "clang diagnostic ignored \"-Wparentheses\"" )

#    define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
         _Pragma( "clang diagnostic ignored \"-Wunused-variable\"" )

#    define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
         _Pragma( "clang diagnostic ignored \"-Wgnu-zero-variadic-macro-arguments\"" )

#    define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
         _Pragma( "clang diagnostic ignored \"-Wunused-template\"" )

#endif // __clang__

// Assume that non-Windows platforms support posix signals by default
#if !defined(CATCH_PLATFORM_WINDOWS)
    #define CATCH_INTERNAL_CONFIG_POSIX_SIGNALS
#endif

// We know some environments not to support full POSIX signals
#if defined(__CYGWIN__) || defined(__QNX__) || defined(__EMSCRIPTEN__) || defined(__DJGPP__)
    #define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
#endif

#ifdef __OS400__
#       define CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS
#       define CATCH_CONFIG_COLOUR_NONE
#endif

// Android somehow still does not support std::to_string
#if defined(__ANDROID__)
#    define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING
#    define CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE
#endif

// Not all Windows environments support SEH properly
#if defined(__MINGW32__)
#    define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
#endif

// PS4
#if defined(__ORBIS__)
#    define CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE
#endif

// Cygwin
#ifdef __CYGWIN__

// Required for some versions of Cygwin to declare gettimeofday
// see: http://stackoverflow.com/questions/36901803/gettimeofday-not-declared-in-this-scope-cygwin
#   define _BSD_SOURCE
// some versions of cygwin (most) do not support std::to_string. Use the libstd check.
// https://gcc.gnu.org/onlinedocs/gcc-4.8.2/libstdc++/api/a01053_source.html line 2812-2813
# if !((__cplusplus >= 201103L) && defined(_GLIBCXX_USE_C99) \
           && !defined(_GLIBCXX_HAVE_BROKEN_VSWPRINTF))

#    define CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING

```

```
# endif
#endif // __CYGWIN__

// Visual C++
#if defined(_MSC_VER)

// Universal Windows platform does not support SEH
// Or console colours (or console at all...)
#  if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
#    define CATCH_CONFIG_COLOUR_NONE
#  else
#    define CATCH_INTERNAL_CONFIG_WINDOWS_SEH
#  endif

#  if !defined(__clang__) // Handle Clang masquerading for msvc

// MSVC traditional preprocessor needs some workaround for __VA_ARGS__
// _MSVC_TRADITIONAL == 0 means new conformant preprocessor
// _MSVC_TRADITIONAL == 1 means old traditional non-conformant preprocessor
#    if !defined(_MSVC_TRADITIONAL) || (defined(_MSVC_TRADITIONAL) && _MSVC_TRADITIONAL)
#      define CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#    endif // MSVC_TRADITIONAL

// Only do this if we're not using clang on Windows, which uses `diagnostic push` & `diagnostic pop`
#    define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION __pragma( warning(push) )
#    define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION  __pragma( warning(pop) )
#  endif // __clang__

#endif // _MSC_VER

#if defined(_REENTRANT) || defined(_MSC_VER)
// Enable async processing, as -pthread is specified or no additional linking is required
# define CATCH_INTERNAL_CONFIG_USE_ASYNC
#endif // _MSC_VER

// Check if we are compiled with -fno-exceptions or equivalent
#if defined(__EXCEPTIONS) || defined(__cpp_exceptions) || defined(_CPPUNWIND)
#  define CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED
#endif

// DJGPP
#ifdef __DJGPP__
#  define CATCH_INTERNAL_CONFIG_NO_WCHAR
#endif // __DJGPP__

// Embarcadero C++Build
#if defined(__BORLANDC__)
    #define CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN
#endif


// Use of __COUNTER__ is suppressed during code analysis in
// CLion/AppCode 2017.2.x and former, because __COUNTER__ is not properly
// handled by it.
// Otherwise all supported compilers support COUNTER macro,
// but user still might want to turn it off
#if ( !defined(__JETBRAINS_IDE__) || __JETBRAINS_IDE__ >= 20170300L )
    #define CATCH_INTERNAL_CONFIG_COUNTER
#endif


// RTX is a special version of Windows that is real time.
// This means that it is detected as Windows, but does not provide
// the same set of capabilities as real Windows does.
#if defined(UNDER_RTSS) || defined(RTX64_BUILD)
    #define CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH
    #define CATCH_INTERNAL_CONFIG_NO_ASYNC
    #define CATCH_CONFIG_COLOUR_NONE
#endif

#if !defined(_GLIBCXX_USE_C99_MATH_TR1)
#define CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER
#endif

// Various stdlib support checks that require __has_include
#if defined(__has_include)
  // Check if string_view is available and usable
  #if __has_include(<string_view>) && defined(CATCH_CPP17_OR_GREATER)
  #    define CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW
  #endif

  // Check if optional is available and usable
  #  if __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)
  #    define CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL
  #  endif // __has_include(<optional>) && defined(CATCH_CPP17_OR_GREATER)

  // Check if byte is available and usable
  #  if __has_include(<cstddef>) && defined(CATCH_CPP17_OR_GREATER)
  #    include <cstddef>
  #    if defined(__cpp_lib_byte) && (__cpp_lib_byte > 0)
  #      define CATCH_INTERNAL_CONFIG_CPP17_BYTE
  #    endif
  #  endif // __has_include(<cstddef>) && defined(CATCH_CPP17_OR_GREATER)

  // Check if variant is available and usable
  #  if __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
  #    if defined(__clang__) && (__clang_major__ < 8)
         // work around clang bug with libstdc++ https://bugs.llvm.org/show_bug.cgi?id=31852
         // fix should be in clang 8, workaround in libstdc++ 8.2
  #      include <ciso646>
  #      if defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
  #        define CATCH_CONFIG_NO_CPP17_VARIANT
  #      else
  #        define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
  #      endif // defined(__GLIBCXX__) && defined(_GLIBCXX_RELEASE) && (_GLIBCXX_RELEASE < 9)
  #    else
  #      define CATCH_INTERNAL_CONFIG_CPP17_VARIANT
  #    endif // defined(__clang__) && (__clang_major__ < 8)
  #  endif // __has_include(<variant>) && defined(CATCH_CPP17_OR_GREATER)
#endif // defined(__has_include)

#if defined(CATCH_INTERNAL_CONFIG_COUNTER) && !defined(CATCH_CONFIG_NO_COUNTER) &&
!defined(CATCH_CONFIG_COUNTER)
#   define CATCH_CONFIG_COUNTER
#endif
#if defined(CATCH_INTERNAL_CONFIG_WINDOWS_SEH) && !defined(CATCH_CONFIG_NO_WINDOWS_SEH) &&
!defined(CATCH_CONFIG_WINDOWS_SEH) && !defined(CATCH_INTERNAL_CONFIG_NO_WINDOWS_SEH)
#   define CATCH_CONFIG_WINDOWS_SEH
#endif
// This is set by default, because we assume that unix compilers are posix-signal-compatible by
default.
#if defined(CATCH_INTERNAL_CONFIG_POSIX_SIGNALS) && !defined(CATCH_INTERNAL_CONFIG_NO_POSIX_SIGNALS)
&& !defined(CATCH_CONFIG_NO_POSIX_SIGNALS) && !defined(CATCH_CONFIG_POSIX_SIGNALS)
#   define CATCH_CONFIG_POSIX_SIGNALS
#endif
// This is set by default, because we assume that compilers with no wchar_t support are just rare
exceptions.
#if !defined(CATCH_INTERNAL_CONFIG_NO_WCHAR) && !defined(CATCH_CONFIG_NO_WCHAR) &&
!defined(CATCH_CONFIG_WCHAR)
#   define CATCH_CONFIG_WCHAR
#endif

#if !defined(CATCH_INTERNAL_CONFIG_NO_CPP11_TO_STRING) && !defined(CATCH_CONFIG_NO_CPP11_TO_STRING) &&
!defined(CATCH_CONFIG_CPP11_TO_STRING)
#    define CATCH_CONFIG_CPP11_TO_STRING
#endif

#if defined(CATCH_INTERNAL_CONFIG_CPP17_OPTIONAL) && !defined(CATCH_CONFIG_NO_CPP17_OPTIONAL) &&
!defined(CATCH_CONFIG_CPP17_OPTIONAL)
#  define CATCH_CONFIG_CPP17_OPTIONAL
#endif

#if defined(CATCH_INTERNAL_CONFIG_CPP17_STRING_VIEW) && !defined(CATCH_CONFIG_NO_CPP17_STRING_VIEW) &&
!defined(CATCH_CONFIG_CPP17_STRING_VIEW)
#  define CATCH_CONFIG_CPP17_STRING_VIEW
#endif

#if defined(CATCH_INTERNAL_CONFIG_CPP17_VARIANT) && !defined(CATCH_CONFIG_NO_CPP17_VARIANT) &&
!defined(CATCH_CONFIG_CPP17_VARIANT)
#  define CATCH_CONFIG_CPP17_VARIANT
#endif

#if defined(CATCH_INTERNAL_CONFIG_CPP17_BYTE) && !defined(CATCH_CONFIG_NO_CPP17_BYTE) &&
!defined(CATCH_CONFIG_CPP17_BYTE)
#  define CATCH_CONFIG_CPP17_BYTE
#endif

#if defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
#  define CATCH_INTERNAL_CONFIG_NEW_CAPTURE
#endif

#if defined(CATCH_INTERNAL_CONFIG_NEW_CAPTURE) && !defined(CATCH_INTERNAL_CONFIG_NO_NEW_CAPTURE) &&
!defined(CATCH_CONFIG_NO_NEW_CAPTURE) && !defined(CATCH_CONFIG_NEW_CAPTURE)
#  define CATCH_CONFIG_NEW_CAPTURE
#endif

#if !defined(CATCH_INTERNAL_CONFIG_EXCEPTIONS_ENABLED) && !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
#  define CATCH_CONFIG_DISABLE_EXCEPTIONS
#endif

#if defined(CATCH_INTERNAL_CONFIG_POLYFILL_ISNAN) && !defined(CATCH_CONFIG_NO_POLYFILL_ISNAN) &&
!defined(CATCH_CONFIG_POLYFILL_ISNAN)
#  define CATCH_CONFIG_POLYFILL_ISNAN
#endif
```

```

#if defined(CATCH_INTERNAL_CONFIG_USE_ASYNC)  && !defined(CATCH_INTERNAL_CONFIG_NO_ASYNC) && !defined(CATCH_CONFIG_NO_USE_ASYNC) && !defined(CATCH_CONFIG_USE_ASYNC)
#  define CATCH_CONFIG_USE_ASYNC
#endif

#if defined(CATCH_INTERNAL_CONFIG_ANDROID_LOGWRITE) && !defined(CATCH_CONFIG_NO_ANDROID_LOGWRITE) && !defined(CATCH_CONFIG_ANDROID_LOGWRITE)
#  define CATCH_CONFIG_ANDROID_LOGWRITE
#endif

#if defined(CATCH_INTERNAL_CONFIG_GLOBAL_NEXTAFTER) && !defined(CATCH_CONFIG_NO_GLOBAL_NEXTAFTER) && !defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
#  define CATCH_CONFIG_GLOBAL_NEXTAFTER
#endif

// Even if we do not think the compiler has that warning, we still have
// to provide a macro that can be used by the code.
#if !defined(CATCH_INTERNAL_START_WARNINGS_SUPPRESSION)
#   define CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
#endif
#if !defined(CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION)
#   define CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
#endif
#if !defined(CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS)
#   define CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS
#endif
#if !defined(CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS)
#   define CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
#endif
#if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS)
#   define CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS
#endif
#if !defined(CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS)
#   define CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS
#endif

// The goal of this macro is to avoid evaluation of the arguments, but
// still have the compiler warn on problems inside...
#if !defined(CATCH_INTERNAL_IGNORE_BUT_WARN)
#   define CATCH_INTERNAL_IGNORE_BUT_WARN(...)
#endif

#if defined(__APPLE__) && defined(__apple_build_version__) && (__clang_major__ < 10)
#   undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
#elif defined(__clang__) && (__clang_major__ < 5)
#   undef CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
#endif

#if !defined(CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS)
#   define CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS
#endif

#if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
#define CATCH_TRY if ((true))
#define CATCH_CATCH_ALL if ((false))
#define CATCH_CATCH_ANON(type) if ((false))
#else
#define CATCH_TRY try
#define CATCH_CATCH_ALL catch (...)
#define CATCH_CATCH_ANON(type) catch (type)
#endif

#if defined(CATCH_INTERNAL_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR) && !defined(CATCH_CONFIG_NO_TRADITIONAL_MSVC_PREPROCESSOR) && !defined(CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR)
#define CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#endif

// end catch_compiler_capabilities.h
#define INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line ) name##line
#define INTERNAL_CATCH_UNIQUE_NAME_LINE( name, line ) INTERNAL_CATCH_UNIQUE_NAME_LINE2( name, line )
#ifdef CATCH_CONFIG_COUNTER
#  define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __COUNTER__ )
#else
#  define INTERNAL_CATCH_UNIQUE_NAME( name ) INTERNAL_CATCH_UNIQUE_NAME_LINE( name, __LINE__ )
#endif

#include <iosfwd>
#include <string>
#include <cstdint>

// We need a dummy global operator« so we can bring it into Catch namespace later
struct Catch_global_namespace_dummy {};
std::ostream& operator«(std::ostream&, Catch_global_namespace_dummy);

namespace Catch {

    struct CaseSensitive { enum Choice {
        Yes,
        No
    }; };

    class NonCopyable {
        NonCopyable( NonCopyable const& )              = delete;
        NonCopyable( NonCopyable && )                  = delete;
        NonCopyable& operator = ( NonCopyable const& ) = delete;
        NonCopyable& operator = ( NonCopyable && )     = delete;

    protected:
        NonCopyable();
        virtual ~NonCopyable();
    };

    struct SourceLineInfo {

        SourceLineInfo() = delete;
        SourceLineInfo( char const* _file, std::size_t _line ) noexcept
        :   file( _file ),
            line( _line )
        {}

        SourceLineInfo( SourceLineInfo const& other )            = default;
        SourceLineInfo& operator = ( SourceLineInfo const& )     = default;
        SourceLineInfo( SourceLineInfo&& )              noexcept = default;
        SourceLineInfo& operator = ( SourceLineInfo&& ) noexcept = default;

        bool empty() const noexcept { return file[0] == '\0'; }
        bool operator == ( SourceLineInfo const& other ) const noexcept;
        bool operator < ( SourceLineInfo const& other ) const noexcept;

        char const* file;
        std::size_t line;
    };

    std::ostream& operator « ( std::ostream& os, SourceLineInfo const& info );

    // Bring in operator« from global namespace into Catch namespace
    // This is necessary because the overload of operator« above makes
    // lookup stop at namespace Catch
    using ::operator«;

    // Use this in variadic streaming macros to allow
    //    » +StreamEndStop
    // as well as
    //    » stuff +StreamEndStop
    struct StreamEndStop {
        std::string operator+() const;
    };
    template<typename T>
    T const& operator + ( T const& value, StreamEndStop ) {
        return value;
    }
}

#define CATCH_INTERNAL_LINEINFO \
    ::Catch::SourceLineInfo( __FILE__, static_cast<std::size_t>( __LINE__ ) )

// end catch_common.h
namespace Catch {

    struct RegistrarForTagAliases {
        RegistrarForTagAliases( char const* alias, char const* tag, SourceLineInfo const& lineInfo );
    };

} // end namespace Catch

#define CATCH_REGISTER_TAG_ALIAS( alias, spec ) \
    CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
    CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
    namespace{ Catch::RegistrarForTagAliases INTERNAL_CATCH_UNIQUE_NAME( AutoRegisterTagAlias )(
alias, spec, CATCH_INTERNAL_LINEINFO ); } \
    CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

// end catch_tag_alias_autoregistrar.h
// start catch_test_registry.h

// start catch_interfaces_testcase.h

#include <vector>

namespace Catch {

    class TestSpec;

    struct ITestInvoker {
        virtual void invoke () const = 0;
        virtual ~ITestInvoker();
    };

    class TestCase;
    struct IConfig;

    struct ITestCaseRegistry {
        virtual ~ITestCaseRegistry();
        virtual std::vector<TestCase> const& getAllTests() const = 0;
        virtual std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const = 0;
    };

    bool isThrowSafe( TestCase const& testCase, IConfig const& config );
    bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );
    std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config );
    std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );

}

// end catch_interfaces_testcase.h
// start catch_stringref.h

#include <cstddef>
#include <string>
#include <iosfwd>
#include <cassert>

namespace Catch {

    class StringRef {
    public:
        using size_type = std::size_t;
        using const_iterator = const char*;

    private:
        static constexpr char const* const s_empty = "";

        char const* m_start = s_empty;
        size_type m_size = 0;

    public: // construction
        constexpr StringRef() noexcept = default;

        StringRef( char const* rawChars ) noexcept;

        constexpr StringRef( char const* rawChars, size_type size ) noexcept
        :   m_start( rawChars ),
            m_size( size )
        {}

        StringRef( std::string const& stdString ) noexcept
        :   m_start( stdString.c_str() ),
            m_size( stdString.size() )
        {}

        explicit operator std::string() const {
            return std::string(m_start, m_size);
        }

    public: // operators
        auto operator == ( StringRef const& other ) const noexcept -> bool;
        auto operator != (StringRef const& other) const noexcept -> bool {
            return !(*this == other);
        }

        auto operator[] ( size_type index ) const noexcept -> char {
            assert(index < m_size);
            return m_start[index];
        }

    public: // named queries
        constexpr auto empty() const noexcept -> bool {
            return m_size == 0;
        }
        constexpr auto size() const noexcept -> size_type {
            return m_size;
        }

        // Returns the current start pointer. If the StringRef is not
        // null-terminated, throws std::domain_exception
        auto c_str() const -> char const*;

    public: // substrings and searches
```

```
        // Returns a substring of [start, start + length).
        // If start + length > size(), then the substring is [start, size()).
        // If start > size(), then the substring is empty.
        auto substr( size_type start, size_type length ) const noexcept -> StringRef;

        // Returns the current start pointer. May not be null-terminated.
        auto data() const noexcept -> char const*;

        constexpr auto isNullTerminated() const noexcept -> bool {
            return m_start[m_size] == ’\0’;
        }

    public: // iterators
        constexpr const_iterator begin() const { return m_start; }
        constexpr const_iterator end() const { return m_start + m_size; }
    };

    auto operator += ( std::string& lhs, StringRef const& sr ) -> std::string&;
    auto operator « ( std::ostream& os, StringRef const& sr ) -> std::ostream&;

    constexpr auto operator "" _sr( char const* rawChars, std::size_t size ) noexcept -> StringRef {
        return StringRef( rawChars, size );
    }
} // namespace Catch

constexpr auto operator "" _catch_sr( char const* rawChars, std::size_t size ) noexcept -> Catch::StringRef {
    return Catch::StringRef( rawChars, size );
}

// end catch_stringref.h
// start catch_preprocessor.hpp


#define CATCH_RECURSION_LEVEL0(...) __VA_ARGS__
#define CATCH_RECURSION_LEVEL1(...) CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(CATCH_RECURSION_LEVEL0(__VA_ARGS__)))
#define CATCH_RECURSION_LEVEL2(...) CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(CATCH_RECURSION_LEVEL1(__VA_ARGS__)))
#define CATCH_RECURSION_LEVEL3(...) CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(CATCH_RECURSION_LEVEL2(__VA_ARGS__)))
#define CATCH_RECURSION_LEVEL4(...) CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(CATCH_RECURSION_LEVEL3(__VA_ARGS__)))
#define CATCH_RECURSION_LEVEL5(...) CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(CATCH_RECURSION_LEVEL4(__VA_ARGS__)))

#ifdef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define INTERNAL_CATCH_EXPAND_VARGS(...) __VA_ARGS__
// MSVC needs more evaluations
#define CATCH_RECURSION_LEVEL6(...) CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(CATCH_RECURSION_LEVEL5(__VA_ARGS__)))
#define CATCH_RECURSE(...)  CATCH_RECURSION_LEVEL6(CATCH_RECURSION_LEVEL6(__VA_ARGS__))
#else
#define CATCH_RECURSE(...)  CATCH_RECURSION_LEVEL5(__VA_ARGS__)
#endif

#define CATCH_REC_END(...)
#define CATCH_REC_OUT

#define CATCH_EMPTY()
#define CATCH_DEFER(id) id CATCH_EMPTY()

#define CATCH_REC_GET_END2() 0, CATCH_REC_END
#define CATCH_REC_GET_END1(...) CATCH_REC_GET_END2
#define CATCH_REC_GET_END(...) CATCH_REC_GET_END1
#define CATCH_REC_NEXT0(test, next, ...) next CATCH_REC_OUT
#define CATCH_REC_NEXT1(test, next) CATCH_DEFER ( CATCH_REC_NEXT0 ) ( test, next, 0)
#define CATCH_REC_NEXT(test, next)  CATCH_REC_NEXT1(CATCH_REC_GET_END test, next)

#define CATCH_REC_LIST0(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) ) ( f, peek, __VA_ARGS__ )
#define CATCH_REC_LIST1(f, x, peek, ...) , f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0) ) ( f, peek, __VA_ARGS__ )
#define CATCH_REC_LIST2(f, x, peek, ...)   f(x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1) ) ( f, peek, __VA_ARGS__ )

#define CATCH_REC_LIST0_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )
#define CATCH_REC_LIST1_UD(f, userdata, x, peek, ...) , f(userdata, x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST0_UD) ) ( f, userdata, peek, __VA_ARGS__ )
#define CATCH_REC_LIST2_UD(f, userdata, x, peek, ...)   f(userdata, x) CATCH_DEFER ( CATCH_REC_NEXT(peek, CATCH_REC_LIST1_UD) ) ( f, userdata, peek, __VA_ARGS__ )

// Applies the function macro `f` to each of the remaining parameters, inserts commas between the results,
// and passes userdata as the first parameter to each invocation,
// e.g. CATCH_REC_LIST_UD(f, x, a, b, c) evaluates to f(x, a), f(x, b), f(x, c)
```

```
#define CATCH_REC_LIST_UD(f, userdata, ...) CATCH_RECURSE(CATCH_REC_LIST2_UD(f, userdata, __VA_ARGS__, ()()(), ()()(), ()()(), 0))

#define CATCH_REC_LIST(f, ...) CATCH_RECURSE(CATCH_REC_LIST2(f, __VA_ARGS__, ()()(), ()()(), ()()(), 0))

#define INTERNAL_CATCH_EXPAND1(param) INTERNAL_CATCH_EXPAND2(param)
#define INTERNAL_CATCH_EXPAND2(...) INTERNAL_CATCH_NO## __VA_ARGS__
#define INTERNAL_CATCH_DEF(...) INTERNAL_CATCH_DEF __VA_ARGS__
#define INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
#define INTERNAL_CATCH_STRINGIZE(...) INTERNAL_CATCH_STRINGIZE2(__VA_ARGS__)
#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define INTERNAL_CATCH_STRINGIZE2(...) #__VA_ARGS__
#define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param) INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param))
#else
// MSVC is adding extra space and needs another indirection to expand INTERNAL_CATCH_NOINTERNAL_CATCH_DEF
#define INTERNAL_CATCH_STRINGIZE2(...) INTERNAL_CATCH_STRINGIZE3(__VA_ARGS__)
#define INTERNAL_CATCH_STRINGIZE3(...) #__VA_ARGS__
#define INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS(param) (INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_REMOVE_PARENS(param)) + 1)
#endif

#define INTERNAL_CATCH_MAKE_NAMESPACE2(...) ns_##__VA_ARGS__
#define INTERNAL_CATCH_MAKE_NAMESPACE(name) INTERNAL_CATCH_MAKE_NAMESPACE2(name)

#define INTERNAL_CATCH_REMOVE_PARENS(...) INTERNAL_CATCH_EXPAND1(INTERNAL_CATCH_DEF __VA_ARGS__)

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define INTERNAL_CATCH_MAKE_TYPE_LIST2(...) decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>())
#define INTERNAL_CATCH_MAKE_TYPE_LIST(...) INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__))
#else
#define INTERNAL_CATCH_MAKE_TYPE_LIST2(...) INTERNAL_CATCH_EXPAND_VARGS(decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS_GEN(__VA_ARGS__)>()))
#define INTERNAL_CATCH_MAKE_TYPE_LIST(...) INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_MAKE_TYPE_LIST2(INTERNAL_CATCH_REMOVE_PARENS(__VA_ARGS__)))
#endif

#define INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(...)\
    CATCH_REC_LIST(INTERNAL_CATCH_MAKE_TYPE_LIST,__VA_ARGS__)

#define INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_0) INTERNAL_CATCH_REMOVE_PARENS(_0)
#define INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_0, _1) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_1_ARG(_1)
#define INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_0, _1, _2) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_2_ARG(_1, _2)
#define INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_0, _1, _2, _3) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_3_ARG(_1, _2, _3)
#define INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_0, _1, _2, _3, _4) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_4_ARG(_1, _2, _3, _4)
#define INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_0, _1, _2, _3, _4, _5) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_5_ARG(_1, _2, _3, _4, _5)
#define INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_0, _1, _2, _3, _4, _5, _6) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_6_ARG(_1, _2, _3, _4, _5, _6)
#define INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_0, _1, _2, _3, _4, _5, _6, _7) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_7_ARG(_1, _2, _3, _4, _5, _6, _7)
#define INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_8_ARG(_1, _2, _3, _4, _5, _6, _7, _8)
#define INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_9_ARG(_1, _2, _3, _4, _5, _6, _7, _8, _9)
#define INTERNAL_CATCH_REMOVE_PARENS_11_ARG(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10) INTERNAL_CATCH_REMOVE_PARENS(_0), INTERNAL_CATCH_REMOVE_PARENS_10_ARG(_1, _2, _3, _4, _5, _6, _7, _8, _9, _10)

#define INTERNAL_CATCH_VA_NARGS_IMPL(_0, _1, _2, _3, _4, _5, _6, _7, _8, _9, _10, N, ...) N

#define INTERNAL_CATCH_TYPE_GEN\
    template<typename...> struct TypeList {};\
    template<typename...Ts>\
    constexpr auto get_wrapper() noexcept -> TypeList<Ts...> { return {}; }\
    template<template<typename...> class...> struct TemplateTypeList{};\
    template<template<typename...> class...Cs>\
    constexpr auto get_wrapper() noexcept -> TemplateTypeList<Cs...> { return {}; }\
    template<typename...>\
    struct append;\
    template<typename...>\
    struct rewrap;\
    template<template<typename...> class, typename...>\
    struct create;\
    template<template<typename...> class, typename>\
    struct convert;\
    \
    template<typename T> \
    struct append<T> { using type = T; };\
    template< template<typename...> class L1, typename...E1, template<typename...> class L2, typename...E2, typename...Rest>\
    struct append<L1<E1...>, L2<E2...>, Rest...> { using type = typename append<L1<E1...,E2...>, Rest...>::type; };\
    template< template<typename...> class L1, typename...E1, typename...Rest>\
    struct append<L1<E1...>, TypeList<mpl_::na>, Rest...> { using type = L1<E1...>; };\
    \
    template< template<typename...> class Container, template<typename...> class List, typename...elems>\
    struct rewrap<TemplateTypeList<Container>, List<elems...» { using type = TypeList<Container<elems...»; };\
    template< template<typename...> class Container, template<typename...> class List, class...Elems, typename...Elements>\
    struct rewrap<TemplateTypeList<Container>, List<Elems...>, Elements...> { using type = typename append<TypeList<Container<Elems...», typename rewrap<TemplateTypeList<Container>, Elements...>::type>::type; };\
    \
    template<template <typename...> class Final, template< typename...> class...Containers, typename...Types>\
    struct create<Final, TemplateTypeList<Containers...>, TypeList<Types...» { using type = typename append<Final<>, typename rewrap<TemplateTypeList<Containers>, Types...>::type...>::type; };\
    template<template <typename...> class Final, template <typename...> class List, typename...Ts>\
    struct convert<Final, List<Ts...» { using type = typename append<Final<>,TypeList<Ts>...>::type; };

#define INTERNAL_CATCH_NTTP_1(signature, ...)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)> struct Nttp{};\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    constexpr auto get_wrapper() noexcept -> Nttp<__VA_ARGS__> { return {}; } \
    template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...> struct NttpTemplateTypeList{};\
    template<template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...Cs>\
    constexpr auto get_wrapper() noexcept -> NttpTemplateTypeList<Cs...> { return {}; } \
    \
    template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container, template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List, INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__» { using type = TypeList<Container<__VA_ARGS__»; };\
    template< template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class Container, template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class List, INTERNAL_CATCH_REMOVE_PARENS(signature), typename...Elements>\
    struct rewrap<NttpTemplateTypeList<Container>, List<__VA_ARGS__>, Elements...> { using type = typename append<TypeList<Container<__VA_ARGS__», typename rewrap<NttpTemplateTypeList<Container>, Elements...>::type>::type; };\
    template<template <typename...> class Final, template<INTERNAL_CATCH_REMOVE_PARENS(signature)> class...Containers, typename...Types>\
    struct create<Final, NttpTemplateTypeList<Containers...>, TypeList<Types...» { using type = typename append<Final<>, typename rewrap<NttpTemplateTypeList<Containers>, Types...>::type...>::type; };

#define INTERNAL_CATCH_DECLARE_SIG_TEST0(TestName)
#define INTERNAL_CATCH_DECLARE_SIG_TEST1(TestName, signature)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    static void TestName()
#define INTERNAL_CATCH_DECLARE_SIG_TEST_X(TestName, signature, ...)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    static void TestName()

#define INTERNAL_CATCH_DEFINE_SIG_TEST0(TestName)
#define INTERNAL_CATCH_DEFINE_SIG_TEST1(TestName, signature)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    static void TestName()
#define INTERNAL_CATCH_DEFINE_SIG_TEST_X(TestName, signature,...)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    static void TestName()

#define INTERNAL_CATCH_NTTP_REGISTER0(TestFunc, signature)\
    template<typename Type>\
    void reg_test(TypeList<Type>, Catch::NameAndTags nameAndTags)\
    {\
        Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<Type>), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), nameAndTags);\
    }

#define INTERNAL_CATCH_NTTP_REGISTER(TestFunc, signature, ...)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    void reg_test(Nttp<__VA_ARGS__>, Catch::NameAndTags nameAndTags)\
    {\
        Catch::AutoReg( Catch::makeTestInvoker(&TestFunc<__VA_ARGS__>), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), nameAndTags);\
    }

#define INTERNAL_CATCH_NTTP_REGISTER_METHOD0(TestName, signature, ...)\
    template<typename Type>\
    void reg_test(TypeList<Type>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\
    {\
        Catch::AutoReg( Catch::makeTestInvoker(&TestName<Type>::test), CATCH_INTERNAL_LINEINFO,
className, nameAndTags);\
    }

#define INTERNAL_CATCH_NTTP_REGISTER_METHOD(TestName, signature, ...)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)>\
    void reg_test(Nttp<__VA_ARGS__>, Catch::StringRef className, Catch::NameAndTags nameAndTags)\
    {\
        Catch::AutoReg( Catch::makeTestInvoker(&TestName<__VA_ARGS__>::test), CATCH_INTERNAL_LINEINFO,
className, nameAndTags);\
    }

#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0(TestName, ClassName)
#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1(TestName, ClassName, signature)\
    template<typename TestType> \
    struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<TestType> { \
        void test();\
    }

#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X(TestName, ClassName, signature, ...)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
    struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName)<__VA_ARGS__> { \
        void test();\
    }

#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0(TestName)
#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1(TestName, signature)\
    template<typename TestType> \
    void INTERNAL_CATCH_MAKE_NAMESPACE(TestName)::TestName<TestType>::test()
#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X(TestName, signature, ...)\
    template<INTERNAL_CATCH_REMOVE_PARENS(signature)> \
    void INTERNAL_CATCH_MAKE_NAMESPACE(TestName)::TestName<__VA_ARGS__>::test()

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define INTERNAL_CATCH_NTTP_0
#define INTERNAL_CATCH_NTTP_GEN(...) INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,
INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__),
INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1(__VA_ARGS__),
INTERNAL_CATCH_NTTP_1(__VA_ARGS__), INTERNAL_CATCH_NTTP_1( __VA_ARGS__), INTERNAL_CATCH_NTTP_1(
__VA_ARGS__), INTERNAL_CATCH_NTTP_1( __VA_ARGS__), INTERNAL_CATCH_NTTP_1(
__VA_ARGS__),INTERNAL_CATCH_NTTP_1( __VA_ARGS__), INTERNAL_CATCH_NTTP_0)
#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
__VA_ARGS__, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0)(TestName, __VA_ARGS__)
#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, ...) INTERNAL_CATCH_VA_NARGS_IMPL(
"dummy", __VA_ARGS__,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0)(TestName, ClassName,
__VA_ARGS__)
#define INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
__VA_ARGS__, INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD0,
INTERNAL_CATCH_NTTP_REGISTER_METHOD0)(TestName, __VA_ARGS__)
#define INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER0, INTERNAL_CATCH_NTTP_REGISTER0)(TestFunc, __VA_ARGS__)
#define INTERNAL_CATCH_DEFINE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
__VA_ARGS__, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST1,
INTERNAL_CATCH_DEFINE_SIG_TEST0)(TestName, __VA_ARGS__)
#define INTERNAL_CATCH_DECLARE_SIG_TEST(TestName, ...) INTERNAL_CATCH_VA_NARGS_IMPL( "dummy",
__VA_ARGS__, INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST1, INTERNAL_CATCH_DECLARE_SIG_TEST0)(TestName, __VA_ARGS__)
#define INTERNAL_CATCH_REMOVE_PARENS_GEN(...) INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,
INTERNAL_CATCH_REMOVE_PARENS_11_ARG,INTERNAL_CATCH_REMOVE_PARENS_10_ARG,INTERNAL_CATCH_REMOVE_PARENS_9_ARG,INTERNAL_CATC
#else
#define INTERNAL_CATCH_NTTP_0(signature)
#define INTERNAL_CATCH_NTTP_GEN(...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__, INTERNAL_CATCH_NTTP_1,
INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1,
INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_1,
INTERNAL_CATCH_NTTP_1,INTERNAL_CATCH_NTTP_1, INTERNAL_CATCH_NTTP_0)( __VA_ARGS__))
#define INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD1, INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD0)(TestName,
__VA_ARGS__))
#define INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD_X,INTERNAL_CATCH_DECLARE_SIG_TEST_METHO
INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD1, INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD0)(TestName, ClassName,
__VA_ARGS__))
#define INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD,
INTERNAL_CATCH_NTTP_REGISTER_METHOD, INTERNAL_CATCH_NTTP_REGISTER_METHOD0,
INTERNAL_CATCH_NTTP_REGISTER_METHOD0)(TestName, __VA_ARGS__))
#define INTERNAL_CATCH_NTTP_REG_GEN(TestFunc, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER, INTERNAL_CATCH_NTTP_REGISTER,
INTERNAL_CATCH_NTTP_REGISTER0, INTERNAL_CATCH_NTTP_REGISTER0)(TestFunc, __VA_ARGS__))
#define INTERNAL_CATCH_DEFINE_SIG_TEST(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X, INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DEFINE_SIG_TEST1,
INTERNAL_CATCH_DEFINE_SIG_TEST0)(TestName, __VA_ARGS__))
#define INTERNAL_CATCH_DECLARE_SIG_TEST(TestName, ...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL( "dummy", __VA_ARGS__,
INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST_X, INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DEFINE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,INTERNAL_CATCH_DECLARE_SIG_TEST_X,
INTERNAL_CATCH_DECLARE_SIG_TEST1, INTERNAL_CATCH_DECLARE_SIG_TEST0)(TestName, __VA_ARGS__))
#define INTERNAL_CATCH_REMOVE_PARENS_GEN(...)
INTERNAL_CATCH_EXPAND_VARGS(INTERNAL_CATCH_VA_NARGS_IMPL(__VA_ARGS__,
INTERNAL_CATCH_REMOVE_PARENS_11_ARG,INTERNAL_CATCH_REMOVE_PARENS_10_ARG,INTERNAL_CATCH_REMOVE_PARENS_9_ARG,INTERNAL_CATC
#endif

// end catch_preprocessor.hpp
// start catch_meta.hpp


#include <type_traits>

namespace Catch {
    template<typename T>
    struct always_false : std::false_type {};

    template <typename> struct true_given : std::true_type {};
    struct is_callable_tester {
        template <typename Fun, typename... Args>
        true_given<decltype(std::declval<Fun>()(std::declval<Args>()...))> static test(int);
        template <typename...>
        std::false_type static test(...);
    };

    template <typename T>
    struct is_callable;

    template <typename Fun, typename... Args>
    struct is_callable<Fun(Args...)> : decltype(is_callable_tester::test<Fun, Args...>(0)) {};

#if defined(__cpp_lib_is_invocable) && __cpp_lib_is_invocable >= 201703
    // std::result_of is deprecated in C++17 and removed in C++20. Hence, it is
    // replaced with std::invoke_result here.
    template <typename Func, typename... U>
    using FunctionReturnType = std::remove_reference_t<std::remove_cv_t<std::invoke_result_t<Func,
U...»>;
#else
    // Keep ::type here because we still support C++11
    template <typename Func, typename... U>
    using FunctionReturnType = typename std::remove_reference<typename std::remove_cv<typename
std::result_of<Func(U...)>::type>::type>::type;
#endif
```

```

} // namespace Catch

namespace mpl_{
    struct na;
}

// end catch_meta.hpp
namespace Catch {

template<typename C>
class TestInvokerAsMethod : public ITestInvoker {
    void (C::*m_testAsMethod)();
public:
    TestInvokerAsMethod( void (C::*testAsMethod)() ) noexcept : m_testAsMethod( testAsMethod ) {}

    void invoke() const override {
        C obj;
        (obj.*m_testAsMethod)();
    }
};

auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker*;

template<typename C>
auto makeTestInvoker( void (C::*testAsMethod)() ) noexcept -> ITestInvoker* {
    return new(std::nothrow) TestInvokerAsMethod<C>( testAsMethod );
}

struct NameAndTags {
    NameAndTags( StringRef const& name_ = StringRef(), StringRef const& tags_ = StringRef() ) noexcept;
    StringRef name;
    StringRef tags;
};

struct AutoReg : NonCopyable {
    AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const& classOrMethod, NameAndTags const& nameAndTags ) noexcept;
    ~AutoReg();
};

} // end namespace Catch

#if defined(CATCH_CONFIG_DISABLE)
    #define INTERNAL_CATCH_TESTCASE_NO_REGISTRATION( TestName, ... ) \
        static void TestName()
    #define INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION( TestName, ClassName, ... ) \
        namespace{                        \
            struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
                void test();              \
            };                            \
        }                                 \
        void TestName::test()
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( TestName, TestFunc, Name, Tags, Signature, ... )  \
        INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature))
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( TestNameClass, TestName, ClassName, Name, Tags, Signature, ... )    \
        namespace{                                                                                  \
            namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) {                                      \
            INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName, INTERNAL_CATCH_REMOVE_PARENS(Signature));\
        }                                                                                           \
        }                                                                                           \
        INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))

    #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(Name, Tags, ...) \
            INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename TestType, __VA_ARGS__ )
    #else
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(Name, Tags, ...) \
            INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename TestType, __VA_ARGS__ ) )
    #endif

    #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(Name, Tags, Signature, ...) \
            INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ )
    #else
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(Name, Tags, Signature, ...) \
            INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION_2(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ ) )
    #endif

    #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... ) \
            INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, typename T, __VA_ARGS__ )
    #else
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION( ClassName, Name, Tags,... ) \
            INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, typename T, __VA_ARGS__ ) )
    #endif

    #ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags, Signature, ... ) \
            INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, Signature, __VA_ARGS__ )
    #else
        #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION( ClassName, Name, Tags, Signature, ... ) \
            INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, Signature, __VA_ARGS__ ) )
    #endif
#endif

    #define INTERNAL_CATCH_TESTCASE2( TestName, ... ) \
        static void TestName(); \
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( &TestName ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); } /* NOLINT */ \
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
        static void TestName()
    #define INTERNAL_CATCH_TESTCASE( ... ) \
        INTERNAL_CATCH_TESTCASE2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), __VA_ARGS__ )

    #define INTERNAL_CATCH_METHOD_AS_TEST_CASE( QualifiedMethod, ... ) \
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        namespace{ Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( &QualifiedMethod ), CATCH_INTERNAL_LINEINFO, "&" #QualifiedMethod, Catch::NameAndTags{ __VA_ARGS__ } ); } /* NOLINT */ \
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

    #define INTERNAL_CATCH_TEST_CASE_METHOD2( TestName, ClassName, ... )\
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        namespace{ \
            struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName) { \
                void test(); \
            }; \
            Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar ) ( Catch::makeTestInvoker( &TestName::test ), CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT */ \
        } \
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
        void TestName::test()
    #define INTERNAL_CATCH_TEST_CASE_METHOD( ClassName, ... ) \
        INTERNAL_CATCH_TEST_CASE_METHOD2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), ClassName, __VA_ARGS__ )

    #define INTERNAL_CATCH_REGISTER_TESTCASE( Function, ... ) \
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        Catch::AutoReg INTERNAL_CATCH_UNIQUE_NAME( autoRegistrar )( Catch::makeTestInvoker( Function ), CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ __VA_ARGS__ } ); /* NOLINT */ \
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_2(TestName, TestFunc, Name, Tags, Signature, ... )\
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
        INTERNAL_CATCH_DECLARE_SIG_TEST(TestFunc, INTERNAL_CATCH_REMOVE_PARENS(Signature));\
        namespace {\
        namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){\
            INTERNAL_CATCH_TYPE_GEN\
            INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
            INTERNAL_CATCH_NTTP_REG_GEN(TestFunc,INTERNAL_CATCH_REMOVE_PARENS(Signature))\
            template<typename...Types> \
            struct TestName{\
                TestName(){\
                    int index = 0;                                    \
                    constexpr char const* tmpl_types[] =
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
                    using expander = int[];\
                    (void)expander{(reg_test(Types{}, Catch::NameAndTags{ Name " - " +
std::string(tmpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
                }\
            };\
            static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
            TestName<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
            return 0;\
        }();\
        }\
        }\
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
        INTERNAL_CATCH_DEFINE_SIG_TEST(TestFunc,INTERNAL_CATCH_REMOVE_PARENS(Signature))

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename TestType, __VA_ARGS__ )
#else
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE(Name, Tags, ...) \
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename TestType, __VA_ARGS__ ) )
#endif

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(Name, Tags, Signature, ...) \
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ )
#else
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG(Name, Tags, Signature, ...) \
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ ) )
#endif

    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(TestName, TestFuncName, Name, Tags, Signature,
TmplTypes, TypesList) \
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION                      \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS                      \
        CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS                \
        CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS              \
        template<typename TestType> static void TestFuncName();       \
        namespace {\
        namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName) {                                    \
            INTERNAL_CATCH_TYPE_GEN                                                  \
            INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))         \
            template<typename... Types>                               \
            struct TestName {                                         \
                void reg_tests() {                                          \
                    int index = 0;                                    \
                    using expander = int[];                           \
                    constexpr char const* tmpl_types[] =
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))};\
                    constexpr char const* types_list[] =
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))};\
                    constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]);\
                    (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestFuncName<Types> ),
CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " +
std::string(tmpl_types[index / num_types]) + "<" + std::string(types_list[index % num_types]) + ">",
Tags } ), index++)... };/* NOLINT */\
                }                                                     \
            };                                                        \
            static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
                using TestInit = typename create<TestName,
decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>()),
TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))»::type; \
                TestInit t;                                           \
                t.reg_tests();                                        \
                return 0;                                             \
            }();                                                      \
        }                                                             \
        }                                                             \
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION                       \
        template<typename TestType>                                   \
        static void TestFuncName()
```

```

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...)\
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename T,__VA_ARGS__)
#else
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE(Name, Tags, ...)\
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, typename T, __VA_ARGS__ ) )
#endif

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...)\
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__)
#else
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(Name, Tags, Signature, ...)\
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE2(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, Signature, __VA_ARGS__ ) )
#endif

    #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2(TestName, TestFunc, Name, Tags, TmplList)\
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
        template<typename TestType> static void TestFunc();       \
        namespace {\
        namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){\
        INTERNAL_CATCH_TYPE_GEN\
        template<typename... Types>                               \
        struct TestName {                                         \
            void reg_tests() {                                          \
                int index = 0;                                    \
                using expander = int[];                           \
                (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestFunc<Types> ),
CATCH_INTERNAL_LINEINFO, Catch::StringRef(), Catch::NameAndTags{ Name " - " +
std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)...
};/* NOLINT */\
            }                                                     \
        };\
        static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){ \
                using TestInit = typename convert<TestName, TmplList>::type; \
                TestInit t;                                           \
                t.reg_tests();                                        \
                return 0;                                             \
            }();                                                      \
        }}\
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION                       \
        template<typename TestType>                                   \
        static void TestFunc()

    #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(Name, Tags, TmplList) \
        INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), Name, Tags, TmplList )

    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
Tags, Signature, ... ) \
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
        namespace {\
        namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){ \
            INTERNAL_CATCH_TYPE_GEN\
            INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
            INTERNAL_CATCH_DECLARE_SIG_TEST_METHOD(TestName, ClassName,
INTERNAL_CATCH_REMOVE_PARENS(Signature));\
            INTERNAL_CATCH_NTTP_REG_METHOD_GEN(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))\
            template<typename...Types> \
            struct TestNameClass{\
                TestNameClass(){\
                    int index = 0;                                    \
                    constexpr char const* tmpl_types[] =
{CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, __VA_ARGS__)};\
                    using expander = int[];\
                    (void)expander{(reg_test(Types{}, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(tmpl_types[index]), Tags } ), index++)... };/* NOLINT */ \
                }\
            };\
            static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
                TestNameClass<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(__VA_ARGS__)>();\
                return 0;\
        }();\
        }\
        }\
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
        INTERNAL_CATCH_DEFINE_SIG_TEST_METHOD(TestName, INTERNAL_CATCH_REMOVE_PARENS(Signature))

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, typename T, __VA_ARGS__ )
#else
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( ClassName, Name, Tags,... ) \
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, typename T, __VA_ARGS__ ) )
#endif

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
        INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, Signature, __VA_ARGS__ )
#else
    #define INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature, ... ) \
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_C_L_A_S_S_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ) , ClassName, Name, Tags, Signature, __VA_ARGS__ ) )
#endif

    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2(TestNameClass, TestName, ClassName, Name, Tags, Signature, TmplTypes, TypesList)\
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_ZERO_VARIADIC_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
        template<typename TestType> \
            struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
                void test();\
            };\
        namespace {\
        namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestNameClass) {\
            INTERNAL_CATCH_TYPE_GEN                  \
            INTERNAL_CATCH_NTTP_GEN(INTERNAL_CATCH_REMOVE_PARENS(Signature))\
            template<typename...Types>\
            struct TestNameClass{\
                void reg_tests(){\
                    int index = 0;\
                    using expander = int[];\
                    constexpr char const* tmpl_types[] = {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TmplTypes))};\
                    constexpr char const* types_list[] = {CATCH_REC_LIST(INTERNAL_CATCH_STRINGIZE_WITHOUT_PARENS, INTERNAL_CATCH_REMOVE_PARENS(TypesList))};\
                    constexpr auto num_types = sizeof(types_list) / sizeof(types_list[0]);\
                    (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ), CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " + std::string(tmpl_types[index / num_types]) + "<" + std::string(types_list[index % num_types]) + ">", Tags } ), index++)... };/* NOLINT */ \
                }\
            };\
            static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
                using TestInit = typename create<TestNameClass, decltype(get_wrapper<INTERNAL_CATCH_REMOVE_PARENS(TmplTypes)>()), TypeList<INTERNAL_CATCH_MAKE_TYPE_LISTS_FROM_TYPES(INTERNAL_CATCH_REMOVE_PARENS(TypesList))»::type;\
                TestInit t;\
                t.reg_tests();\
                return 0;\
            }(); \
        }\
        }\
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
        template<typename TestType> \
        void TestName<TestType>::test()

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, typename T, __VA_ARGS__ )
#else
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( ClassName, Name, Tags, ... )\
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, typename T,__VA_ARGS__ ) )
#endif
```

```

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
... )\
        INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, Signature, __VA_ARGS__ )
#else
    #define INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( ClassName, Name, Tags, Signature,
... )\
        INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_2(
INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, Signature,__VA_ARGS__ ) )
#endif

    #define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( TestNameClass, TestName, ClassName, Name,
Tags, TmplList) \
        CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
        CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
        CATCH_INTERNAL_SUPPRESS_UNUSED_TEMPLATE_WARNINGS \
        template<typename TestType> \
        struct TestName : INTERNAL_CATCH_REMOVE_PARENS(ClassName <TestType>) { \
            void test();\
        };\
        namespace {\
        namespace INTERNAL_CATCH_MAKE_NAMESPACE(TestName){ \
            INTERNAL_CATCH_TYPE_GEN\
            template<typename...Types>\
            struct TestNameClass{\
                void reg_tests(){\
                    int index = 0;\
                    using expander = int[];\
                    (void)expander{(Catch::AutoReg( Catch::makeTestInvoker( &TestName<Types>::test ),
CATCH_INTERNAL_LINEINFO, #ClassName, Catch::NameAndTags{ Name " - " +
std::string(INTERNAL_CATCH_STRINGIZE(TmplList)) + " - " + std::to_string(index), Tags } ), index++)...
};/* NOLINT */ \
                }\
            };\
            static int INTERNAL_CATCH_UNIQUE_NAME( globalRegistrar ) = [](){\
                using TestInit = typename convert<TestNameClass, TmplList>::type;\
                TestInit t;\
                t.reg_tests();\
                return 0;\
            }(); \
        }}\
        CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
        template<typename TestType> \
        void TestName<TestType>::test()

#define INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD(ClassName, Name, Tags, TmplList) \
        INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD_2( INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_ ), INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_M_P_L_A_T_E_T_E_S_T_F_U_N_C_ ), ClassName, Name, Tags, TmplList )

// end catch_test_registry.h
// start catch_capture.hpp

// start catch_assertionhandler.h

// start catch_assertioninfo.h

// start catch_result_type.h

namespace Catch {

    // ResultWas::OfType enum
    struct ResultWas { enum OfType {
        Unknown = -1,
        Ok = 0,
        Info = 1,
        Warning = 2,

        FailureBit = 0x10,

        ExpressionFailed = FailureBit | 1,
        ExplicitFailure = FailureBit | 2,

        Exception = 0x100 | FailureBit,

        ThrewException = Exception | 1,
        DidntThrowException = Exception | 2,

        FatalErrorCondition = 0x200 | FailureBit

    }; };

    bool isOk( ResultWas::OfType resultType );
    bool isJustInfo( int flags );

    // ResultDisposition::Flags enum
    struct ResultDisposition { enum Flags {
        Normal = 0x01,

        ContinueOnFailure = 0x02,   // Failures fail test, but execution continues
        FalseTest = 0x04,           // Prefix expression with !
        SuppressFail = 0x08         // Failures are reported but do not fail the test
    }; };

    ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs );

    bool shouldContinueOnFailure( int flags );
    inline bool isFalseTest( int flags ) { return ( flags & ResultDisposition::FalseTest ) != 0; }
    bool shouldSuppressFailure( int flags );

} // end namespace Catch

// end catch_result_type.h
namespace Catch {

    struct AssertionInfo
    {
        StringRef macroName;
        SourceLineInfo lineInfo;
        StringRef capturedExpression;
        ResultDisposition::Flags resultDisposition;

        // We want to delete this constructor but a compiler bug in 4.8 means
        // the struct is then treated as non-aggregate
        //AssertionInfo() = delete;
    };

} // end namespace Catch

// end catch_assertioninfo.h
// start catch_decomposer.h

// start catch_tostring.h

#include <vector>
#include <cstddef>
#include <type_traits>
#include <string>
// start catch_stream.h

#include <iosfwd>
#include <cstddef>
#include <ostream>

namespace Catch {

    std::ostream& cout();
    std::ostream& cerr();
    std::ostream& clog();

    class StringRef;

    struct IStream {
        virtual ~IStream();
        virtual std::ostream& stream() const = 0;
    };

    auto makeStream( StringRef const &filename ) -> IStream const*;

    class ReusableStringStream : NonCopyable {
        std::size_t m_index;
        std::ostream* m_oss;
    public:
        ReusableStringStream();
        ~ReusableStringStream();

        auto str() const -> std::string;

        template<typename T>
        auto operator « ( T const& value ) -> ReusableStringStream& {
            *m_oss « value;
            return *this;
        }
        auto get() -> std::ostream& { return *m_oss; }
    };
}

// end catch_stream.h
// start catch_interfaces_enum_values_registry.h
```

```

#include <vector>

namespace Catch {

    namespace Detail {
        struct EnumInfo {
            StringRef m_name;
            std::vector<std::pair<int, StringRef» m_values;

            ~EnumInfo();

            StringRef lookup( int value ) const;
        };
    } // namespace Detail

    struct IMutableEnumValuesRegistry {
        virtual ~IMutableEnumValuesRegistry();

        virtual Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
std::vector<int> const& values ) = 0;

        template<typename E>
        Detail::EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums,
std::initializer_list<E> values ) {
            static_assert(sizeof(int) >= sizeof(E), "Cannot serialize enum to int");
            std::vector<int> intValues;
            intValues.reserve( values.size() );
            for( auto enumValue : values )
                intValues.push_back( static_cast<int>( enumValue ) );
            return registerEnum( enumName, allEnums, intValues );
        }
    };

} // Catch

// end catch_interfaces_enum_values_registry.h

#ifdef CATCH_CONFIG_CPP17_STRING_VIEW
#include <string_view>
#endif

#ifdef __OBJC__
// start catch_objc_arc.hpp

#import <Foundation/Foundation.h>

#ifdef __has_feature
#define CATCH_ARC_ENABLED __has_feature(objc_arc)
#else
#define CATCH_ARC_ENABLED 0
#endif

void arcSafeRelease( NSObject* obj );
id performOptionalSelector( id obj, SEL sel );

#if !CATCH_ARC_ENABLED
inline void arcSafeRelease( NSObject* obj ) {
    [obj release];
}
inline id performOptionalSelector( id obj, SEL sel ) {
    if( [obj respondsToSelector: sel] )
        return [obj performSelector: sel];
    return nil;
}
#define CATCH_UNSAFE_UNRETAINED
#define CATCH_ARC_STRONG
#else
inline void arcSafeRelease( NSObject* ){}
inline id performOptionalSelector( id obj, SEL sel ) {
#ifdef __clang__
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Warc-performSelector-leaks"
#endif
    if( [obj respondsToSelector: sel] )
        return [obj performSelector: sel];
#ifdef __clang__
#pragma clang diagnostic pop
#endif
    return nil;
}
#define CATCH_UNSAFE_UNRETAINED __unsafe_unretained
#define CATCH_ARC_STRONG __strong
#endif

// end catch_objc_arc.hpp
#endif
```

```

#ifdef _MSC_VER
#pragma warning(push)
#pragma warning(disable:4180) // We attempt to stream a function (address) by const&, which MSVC
complains about but is harmless
#endif

namespace Catch {
    namespace Detail {

        extern const std::string unprintableString;

        std::string rawMemoryToString( const void *object, std::size_t size );

        template<typename T>
        std::string rawMemoryToString( const T& object ) {
          return rawMemoryToString( &object, sizeof(object) );
        }

        template<typename T>
        class IsStreamInsertable {
            template<typename Stream, typename U>
            static auto test(int)
                -> decltype(std::declval<Stream&>() « std::declval<U>(), std::true_type());

            template<typename, typename>
            static auto test(...)->std::false_type;

        public:
            static const bool value = decltype(test<std::ostream, const T&>(0))::value;
        };

        template<typename E>
        std::string convertUnknownEnumToString( E e );

        template<typename T>
        typename std::enable_if<
            !std::is_enum<T>::value && !std::is_base_of<std::exception, T>::value,
        std::string>::type convertUnstreamable( T const& ) {
            return Detail::unprintableString;
        }
        template<typename T>
        typename std::enable_if<
            !std::is_enum<T>::value && std::is_base_of<std::exception, T>::value,
         std::string>::type convertUnstreamable(T const& ex) {
            return ex.what();
        }

        template<typename T>
        typename std::enable_if<
            std::is_enum<T>::value
        , std::string>::type convertUnstreamable( T const& value ) {
            return convertUnknownEnumToString( value );
        }

#if defined(_MANAGED)
        template<typename T>
        std::string clrReferenceToString( T^ ref ) {
            if (ref == nullptr)
                return std::string("null");
            auto bytes = System::Text::Encoding::UTF8->GetBytes(ref->ToString());
            cli::pin_ptr<System::Byte> p = &bytes[0];
            return std::string(reinterpret_cast<char const *>(p), bytes->Length);
        }
#endif

    } // namespace Detail

    // If we decide for C++14, change these to enable_if_ts
    template <typename T, typename = void>
    struct StringMaker {
        template <typename Fake = T>
        static
        typename std::enable_if<::Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
            convert(const Fake& value) {
                ReusableStringStream rss;
                // NB: call using the function-like syntax to avoid ambiguity with
                // user-defined templated operator« under clang.
                rss.operator«(value);
                return rss.str();
        }

        template <typename Fake = T>
        static
        typename std::enable_if<!::Catch::Detail::IsStreamInsertable<Fake>::value, std::string>::type
            convert( const Fake& value ) {
#if !defined(CATCH_CONFIG_FALLBACK_STRINGIFIER)
```

```
            return Detail::convertUnstreamable(value);
#else
            return CATCH_CONFIG_FALLBACK_STRINGIFIER(value);
#endif
        }
    };

    namespace Detail {

        // This function dispatches all stringification requests inside of Catch.
        // Should be preferably called fully qualified, like ::Catch::Detail::stringify
        template <typename T>
        std::string stringify(const T& e) {
            return ::Catch::StringMaker<typename std::remove_cv<typename
std::remove_reference<T>::type>::type>::convert(e);
        }

        template<typename E>
        std::string convertUnknownEnumToString( E e ) {
            return ::Catch::Detail::stringify(static_cast<typename std::underlying_type<E>::type>(e));
        }

#if defined(_MANAGED)
        template <typename T>
        std::string stringify( T^ e ) {
            return ::Catch::StringMaker<T^>::convert(e);
        }
#endif

    } // namespace Detail

    // Some predefined specializations

    template<>
    struct StringMaker<std::string> {
        static std::string convert(const std::string& str);
    };

#ifdef CATCH_CONFIG_CPP17_STRING_VIEW
    template<>
    struct StringMaker<std::string_view> {
        static std::string convert(std::string_view str);
    };
#endif

    template<>
    struct StringMaker<char const *> {
        static std::string convert(char const * str);
    };
    template<>
    struct StringMaker<char *> {
        static std::string convert(char * str);
    };

#ifdef CATCH_CONFIG_WCHAR
    template<>
    struct StringMaker<std::wstring> {
        static std::string convert(const std::wstring& wstr);
    };

# ifdef CATCH_CONFIG_CPP17_STRING_VIEW
    template<>
    struct StringMaker<std::wstring_view> {
        static std::string convert(std::wstring_view str);
    };
# endif

    template<>
    struct StringMaker<wchar_t const *> {
        static std::string convert(wchar_t const * str);
    };
    template<>
    struct StringMaker<wchar_t *> {
        static std::string convert(wchar_t * str);
    };
#endif

    // TBD: Should we use `strnlen` to ensure that we don't go out of the buffer,
    //      while keeping string semantics?
    template<int SZ>
    struct StringMaker<char[SZ]> {
        static std::string convert(char const* str) {
            return ::Catch::Detail::stringify(std::string{ str });
        }
    };
    template<int SZ>
    struct StringMaker<signed char[SZ]> {
        static std::string convert(signed char const* str) {
            return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const *>(str) });
        }
    };
    template<int SZ>
    struct StringMaker<unsigned char[SZ]> {
        static std::string convert(unsigned char const* str) {
            return ::Catch::Detail::stringify(std::string{ reinterpret_cast<char const *>(str) });
        }
    };

#if defined(CATCH_CONFIG_CPP17_BYTE)
    template<>
    struct StringMaker<std::byte> {
        static std::string convert(std::byte value);
    };
#endif // defined(CATCH_CONFIG_CPP17_BYTE)
    template<>
    struct StringMaker<int> {
        static std::string convert(int value);
    };
    template<>
    struct StringMaker<long> {
        static std::string convert(long value);
    };
    template<>
    struct StringMaker<long long> {
        static std::string convert(long long value);
    };
    template<>
    struct StringMaker<unsigned int> {
        static std::string convert(unsigned int value);
    };
    template<>
    struct StringMaker<unsigned long> {
        static std::string convert(unsigned long value);
    };
    template<>
    struct StringMaker<unsigned long long> {
        static std::string convert(unsigned long long value);
    };

    template<>
    struct StringMaker<bool> {
        static std::string convert(bool b);
    };

    template<>
    struct StringMaker<char> {
        static std::string convert(char c);
    };
    template<>
    struct StringMaker<signed char> {
        static std::string convert(signed char c);
    };
    template<>
    struct StringMaker<unsigned char> {
        static std::string convert(unsigned char c);
    };

    template<>
    struct StringMaker<std::nullptr_t> {
        static std::string convert(std::nullptr_t);
    };

    template<>
    struct StringMaker<float> {
        static std::string convert(float value);
        static int precision;
    };

    template<>
    struct StringMaker<double> {
        static std::string convert(double value);
        static int precision;
    };

    template <typename T>
    struct StringMaker<T*> {
        template <typename U>
        static std::string convert(U* p) {
            if (p) {
                return ::Catch::Detail::rawMemoryToString(p);
            } else {
                return "nullptr";
            }
        }
    };

    template <typename R, typename C>
    struct StringMaker<R C::*> {
        static std::string convert(R C::* p) {
            if (p) {
                return ::Catch::Detail::rawMemoryToString(p);
            } else {
                return "nullptr";
            }
        }
    };

#if defined(_MANAGED)
    template <typename T>
    struct StringMaker<T^> {
        static std::string convert( T^ ref ) {
            return ::Catch::Detail::clrReferenceToString(ref);
        }
    };
#endif

    namespace Detail {
        template<typename InputIterator, typename Sentinel = InputIterator>
        std::string rangeToString(InputIterator first, Sentinel last) {
            ReusableStringStream rss;
            rss « "{ ";
            if (first != last) {
                rss « ::Catch::Detail::stringify(*first);
                for (++first; first != last; ++first)
                    rss « ", " « ::Catch::Detail::stringify(*first);
            }
            rss « " }";
            return rss.str();
        }
    }

#ifdef __OBJC__
    template<>
    struct StringMaker<NSString*> {
        static std::string convert(NSString * nsstring) {
            if (!nsstring)
                return "nil";
            return std::string("@") + [nsstring UTF8String];
        }
    };
    template<>
    struct StringMaker<NSObject*> {
        static std::string convert(NSObject* nsObject) {
            return ::Catch::Detail::stringify([nsObject description]);
        }

    };
    namespace Detail {
        inline std::string stringify( NSString* nsstring ) {
            return StringMaker<NSString*>::convert( nsstring );
        }

    } // namespace Detail
#endif // __OBJC__

} // namespace Catch

// Separate std-lib types stringification, so it can be selectively enabled
// This means that we do not bring in

#if defined(CATCH_CONFIG_ENABLE_ALL_STRINGMAKERS)
#  define CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER
#  define CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER
#  define CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER
#  define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
#  define CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER
#endif

// Separate std::pair specialization
#if defined(CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER)
#include <utility>
namespace Catch {
    template<typename T1, typename T2>
    struct StringMaker<std::pair<T1, T2> > {
        static std::string convert(const std::pair<T1, T2>& pair) {
            ReusableStringStream rss;
            rss « "{ "
                « ::Catch::Detail::stringify(pair.first)
                « ", "
                « ::Catch::Detail::stringify(pair.second)
                « " }";
            return rss.str();
        }
    };
}
#endif // CATCH_CONFIG_ENABLE_PAIR_STRINGMAKER

#if defined(CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_OPTIONAL)
#include <optional>
namespace Catch {
    template<typename T>
    struct StringMaker<std::optional<T> > {
        static std::string convert(const std::optional<T>& optional) {
            ReusableStringStream rss;
            if (optional.has_value()) {
                rss « ::Catch::Detail::stringify(*optional);
            } else {
                rss « "{ }";
            }
            return rss.str();
        }
    };
}
#endif // CATCH_CONFIG_ENABLE_OPTIONAL_STRINGMAKER

// Separate std::tuple specialization
#if defined(CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER)
#include <tuple>
namespace Catch {
    namespace Detail {
        template<
            typename Tuple,
            std::size_t N = 0,
            bool = (N < std::tuple_size<Tuple>::value)
            >
            struct TupleElementPrinter {
            static void print(const Tuple& tuple, std::ostream& os) {
                os « (N ? ", " : " ")
                    « ::Catch::Detail::stringify(std::get<N>(tuple));
                TupleElementPrinter<Tuple, N + 1>::print(tuple, os);
            }
        };

        template<
            typename Tuple,
            std::size_t N
        >
            struct TupleElementPrinter<Tuple, N, false> {
            static void print(const Tuple&, std::ostream&) {}
        };

    }

    template<typename ...Types>
    struct StringMaker<std::tuple<Types...» {
        static std::string convert(const std::tuple<Types...>& tuple) {
            ReusableStringStream rss;
            rss « '{';
            Detail::TupleElementPrinter<std::tuple<Types...»::print(tuple, rss.get());
            rss « " }";
            return rss.str();
        }
    };
}
#endif // CATCH_CONFIG_ENABLE_TUPLE_STRINGMAKER

#if defined(CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER) && defined(CATCH_CONFIG_CPP17_VARIANT)
#include <variant>
namespace Catch {
    template<>
    struct StringMaker<std::monostate> {
        static std::string convert(const std::monostate&) {
            return "{ }";
        }
    };

    template<typename... Elements>
    struct StringMaker<std::variant<Elements...» {
        static std::string convert(const std::variant<Elements...>& variant) {
            if (variant.valueless_by_exception()) {
                return "{valueless variant}";
            } else {
                return std::visit(
                    [](const auto& value) {
                        return ::Catch::Detail::stringify(value);
                    },
                    variant
                );
                }
            }
        };
}
#endif // CATCH_CONFIG_ENABLE_VARIANT_STRINGMAKER

namespace Catch {
    // Import begin/ end from std here
    using std::begin;
    using std::end;

    namespace detail {
        template <typename...>
        struct void_type {
            using type = void;
        };

        template <typename T, typename = void>
        struct is_range_impl : std::false_type {
        };

        template <typename T>
        struct is_range_impl<T, typename void_type<decltype(begin(std::declval<T>()))>::type> :
std::true_type {
        };
    } // namespace detail

    template <typename T>
    struct is_range : detail::is_range_impl<T> {
    };

#if defined(_MANAGED) // Managed types are never ranges
    template <typename T>
    struct is_range<T^> {
        static const bool value = false;
    };
#endif

    template<typename Range>
    std::string rangeToString( Range const& range ) {
        return ::Catch::Detail::rangeToString( begin( range ), end( range ) );
    }

    // Handle vector<bool> specially
    template<typename Allocator>
    std::string rangeToString( std::vector<bool, Allocator> const& v ) {
        ReusableStringStream rss;
        rss « "{ ";
        bool first = true;
        for( bool b : v ) {
            if( first )
                first = false;
            else
                rss « ", ";
            rss « ::Catch::Detail::stringify( b );
        }
        rss « " }";
        return rss.str();
    }

    template<typename R>
    struct StringMaker<R, typename std::enable_if<is_range<R>::value &&
!::Catch::Detail::IsStreamInsertable<R>::value>::type> {
        static std::string convert( R const& range ) {
            return rangeToString( range );
        }
    };

    template <typename T, int SZ>
    struct StringMaker<T[SZ]> {
        static std::string convert(T const(&arr)[SZ]) {
            return rangeToString(arr);
        }
    };

} // namespace Catch

// Separate std::chrono::duration specialization
#if defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
#include <ctime>
#include <ratio>
#include <chrono>

namespace Catch {

template <class Ratio>
struct ratio_string {
    static std::string symbol();
};

template <class Ratio>
std::string ratio_string<Ratio>::symbol() {
    Catch::ReusableStringStream rss;
    rss « '[' « Ratio::num « '/'
        « Ratio::den « ']';
    return rss.str();
}
template <>
struct ratio_string<std::atto> {
    static std::string symbol();
};
template <>
struct ratio_string<std::femto> {
    static std::string symbol();
};
template <>
struct ratio_string<std::pico> {
    static std::string symbol();
};
template <>
struct ratio_string<std::nano> {
    static std::string symbol();
};
template <>
struct ratio_string<std::micro> {
    static std::string symbol();
};
template <>
struct ratio_string<std::milli> {
    static std::string symbol();
};

    // std::chrono::duration specializations
    template<typename Value, typename Ratio>
    struct StringMaker<std::chrono::duration<Value, Ratio» {
        static std::string convert(std::chrono::duration<Value, Ratio> const& duration) {
            ReusableStringStream rss;
            rss « duration.count() « ' ' « ratio_string<Ratio>::symbol() « 's';
            return rss.str();
        }
    };
    template<typename Value>
    struct StringMaker<std::chrono::duration<Value, std::ratio<1»> {
        static std::string convert(std::chrono::duration<Value, std::ratio<1» const& duration) {
            ReusableStringStream rss;
            rss « duration.count() « " s";
            return rss.str();
        }
    };
    template<typename Value>
    struct StringMaker<std::chrono::duration<Value, std::ratio<60»> {
        static std::string convert(std::chrono::duration<Value, std::ratio<60» const& duration) {
            ReusableStringStream rss;
            rss « duration.count() « " m";
            return rss.str();
        }
    };
    template<typename Value>
    struct StringMaker<std::chrono::duration<Value, std::ratio<3600»> {
        static std::string convert(std::chrono::duration<Value, std::ratio<3600» const& duration) {
            ReusableStringStream rss;
            rss « duration.count() « " h";
            return rss.str();
        }
    };

    // std::chrono::time_point specialization
    // Generic time_point cannot be specialized, only std::chrono::time_point<system_clock>
    template<typename Clock, typename Duration>
    struct StringMaker<std::chrono::time_point<Clock, Duration» {
        static std::string convert(std::chrono::time_point<Clock, Duration> const& time_point) {
            return ::Catch::Detail::stringify(time_point.time_since_epoch()) + " since epoch";
        }
    };
    // std::chrono::time_point<system_clock> specialization
    template<typename Duration>
    struct StringMaker<std::chrono::time_point<std::chrono::system_clock, Duration» {
        static std::string convert(std::chrono::time_point<std::chrono::system_clock, Duration> const&
time_point) {
            auto converted = std::chrono::system_clock::to_time_t(time_point);

#ifdef _MSC_VER
            std::tm timeInfo = {};
            gmtime_s(&timeInfo, &converted);
```

```
#else
            std::tm* timeInfo = std::gmtime(&converted);
#endif

            auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
            char timeStamp[timeStampSize];
            const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";

#ifdef _MSC_VER
            std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
#else
            std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
#endif
            return std::string(timeStamp);
        }
    };
}
#endif // CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER

#define INTERNAL_CATCH_REGISTER_ENUM( enumName, ... ) \
namespace Catch { \
    template<> struct StringMaker<enumName> { \
        static std::string convert( enumName value ) { \
            static const auto& enumInfo =
::Catch::getMutableRegistryHub().getMutableEnumValuesRegistry().registerEnum( #enumName, #__VA_ARGS__,
{ __VA_ARGS__ } ); \
            return static_cast<std::string>(enumInfo.lookup( static_cast<int>( value ) )); \
        } \
    }; \
}

#define CATCH_REGISTER_ENUM( enumName, ... ) INTERNAL_CATCH_REGISTER_ENUM( enumName, __VA_ARGS__ )

#ifdef _MSC_VER
#pragma warning(pop)
#endif

// end catch_tostring.h
#include <iosfwd>

#ifdef _MSC_VER
#pragma warning(push)
#pragma warning(disable:4389) // '==' : signed/unsigned mismatch
#pragma warning(disable:4018) // more "signed/unsigned mismatch"
#pragma warning(disable:4312) // Converting int to T* using reinterpret_cast (issue on x64 platform)
#pragma warning(disable:4180) // qualifier applied to function type has no meaning
#pragma warning(disable:4800) // Forcing result to true or false
#endif

namespace Catch {

    struct ITransientExpression {
        auto isBinaryExpression() const -> bool { return m_isBinaryExpression; }
        auto getResult() const -> bool { return m_result; }
        virtual void streamReconstructedExpression( std::ostream &os ) const = 0;

        ITransientExpression( bool isBinaryExpression, bool result )
        :   m_isBinaryExpression( isBinaryExpression ),
            m_result( result )
        {}

        // We don't actually need a virtual destructor, but many static analysers
        // complain if it's not here :-(
        virtual ~ITransientExpression();

        bool m_isBinaryExpression;
        bool m_result;

    };

    void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
std::string const& rhs );

    template<typename LhsT, typename RhsT>
    class BinaryExpr  : public ITransientExpression {
        LhsT m_lhs;
        StringRef m_op;
        RhsT m_rhs;

        void streamReconstructedExpression( std::ostream &os ) const override {
            formatReconstructedExpression
                    ( os, Catch::Detail::stringify( m_lhs ), m_op, Catch::Detail::stringify( m_rhs )
);
        }

    public:
        BinaryExpr( bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs )
        :   ITransientExpression{ true, comparisonResult },
            m_lhs( lhs ),
            m_op( op ),
            m_rhs( rhs )
        {}

        template<typename T>
        auto operator && ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename T>
        auto operator || ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename T>
        auto operator == ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename T>
        auto operator != ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename T>
        auto operator > ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename T>
        auto operator < ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename T>
        auto operator >= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename T>
        auto operator <= ( T ) const -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<T>::value,
            "chained comparisons are not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }
    };

    template<typename LhsT>
    class UnaryExpr : public ITransientExpression {
        LhsT m_lhs;

        void streamReconstructedExpression( std::ostream &os ) const override {
            os « Catch::Detail::stringify( m_lhs );
        }

    public:
        explicit UnaryExpr( LhsT lhs )
        :   ITransientExpression{ false, static_cast<bool>(lhs) },
            m_lhs( lhs )
        {}
    };

    // Specialised comparison functions to handle equality comparisons between ints and pointers (NULL
deduces as an int)
    template<typename LhsT, typename RhsT>
    auto compareEqual( LhsT const& lhs, RhsT const& rhs ) -> bool { return static_cast<bool>(lhs ==
rhs); }
    template<typename T>
    auto compareEqual( T* const& lhs, int rhs ) -> bool { return lhs == reinterpret_cast<void const*>(
rhs ); }
    template<typename T>
    auto compareEqual( T* const& lhs, long rhs ) -> bool { return lhs == reinterpret_cast<void
const*>( rhs ); }
    template<typename T>
    auto compareEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }
    template<typename T>
    auto compareEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs )
== rhs; }

    template<typename LhsT, typename RhsT>
    auto compareNotEqual( LhsT const& lhs, RhsT&& rhs ) -> bool { return static_cast<bool>(lhs !=
rhs); }
    template<typename T>
    auto compareNotEqual( T* const& lhs, int rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
    template<typename T>
    auto compareNotEqual( T* const& lhs, long rhs ) -> bool { return lhs != reinterpret_cast<void
const*>( rhs ); }
    template<typename T>
    auto compareNotEqual( int lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>( lhs
) != rhs; }
    template<typename T>
    auto compareNotEqual( long lhs, T* const& rhs ) -> bool { return reinterpret_cast<void const*>(
lhs ) != rhs; }

    template<typename LhsT>
    class ExprLhs {
        LhsT m_lhs;
    public:
        explicit ExprLhs( LhsT lhs ) : m_lhs( lhs ) {}

        template<typename RhsT>
        auto operator == ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
            return { compareEqual( m_lhs, rhs ), m_lhs, "==", rhs };
        }
        auto operator == ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
            return { m_lhs == rhs, m_lhs, "==", rhs };
        }

        template<typename RhsT>
        auto operator != ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
            return { compareNotEqual( m_lhs, rhs ), m_lhs, "!=", rhs };
        }
        auto operator != ( bool rhs ) -> BinaryExpr<LhsT, bool> const {
            return { m_lhs != rhs, m_lhs, "!=", rhs };
        }

        template<typename RhsT>
        auto operator > ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
            return { static_cast<bool>(m_lhs > rhs), m_lhs, ">", rhs };
        }
        template<typename RhsT>
        auto operator < ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
            return { static_cast<bool>(m_lhs < rhs), m_lhs, "<", rhs };
        }
        template<typename RhsT>
        auto operator >= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
            return { static_cast<bool>(m_lhs >= rhs), m_lhs, ">=", rhs };
        }
        template<typename RhsT>
        auto operator <= ( RhsT const& rhs ) -> BinaryExpr<LhsT, RhsT const&> const {
            return { static_cast<bool>(m_lhs <= rhs), m_lhs, "<=", rhs };
        }
        template <typename RhsT>
        auto operator | (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
            return { static_cast<bool>(m_lhs | rhs), m_lhs, "|", rhs };
        }
        template <typename RhsT>
        auto operator & (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
            return { static_cast<bool>(m_lhs & rhs), m_lhs, "&", rhs };
        }
        template <typename RhsT>
        auto operator ^ (RhsT const& rhs) -> BinaryExpr<LhsT, RhsT const&> const {
            return { static_cast<bool>(m_lhs ^ rhs), m_lhs, "^", rhs };
        }

        template<typename RhsT>
        auto operator && ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<RhsT>::value,
            "operator&& is not supported inside assertions, "
            "wrap the expression inside parentheses, or decompose it");
        }

        template<typename RhsT>
        auto operator || ( RhsT const& ) -> BinaryExpr<LhsT, RhsT const&> const {
            static_assert(always_false<RhsT>::value,
            "operator|| is not supported inside assertions, "
```

```
                "wrap the expression inside parentheses, or decompose it");
        }

        auto makeUnaryExpr() const -> UnaryExpr<LhsT> {
            return UnaryExpr<LhsT>{ m_lhs };
        }
    };

    void handleExpression( ITransientExpression const& expr );

    template<typename T>
    void handleExpression( ExprLhs<T> const& expr ) {
        handleExpression( expr.makeUnaryExpr() );
    }

    struct Decomposer {
        template<typename T>
        auto operator <= ( T const& lhs ) -> ExprLhs<T const&> {
            return ExprLhs<T const&>{ lhs };
        }

        auto operator <=( bool value ) -> ExprLhs<bool> {
            return ExprLhs<bool>{ value };
        }
    };

} // end namespace Catch

#ifdef _MSC_VER
#pragma warning(pop)
#endif

// end catch_decomposer.h
// start catch_interfaces_capture.h

#include <string>
#include <chrono>

namespace Catch {

    class AssertionResult;
    struct AssertionInfo;
    struct SectionInfo;
    struct SectionEndInfo;
    struct MessageInfo;
    struct MessageBuilder;
    struct Counts;
    struct AssertionReaction;
    struct SourceLineInfo;

    struct ITransientExpression;
    struct IGeneratorTracker;

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
    struct BenchmarkInfo;
    template <typename Duration = std::chrono::duration<double, std::nano»
    struct BenchmarkStats;
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

    struct IResultCapture {

        virtual ~IResultCapture();

        virtual bool sectionStarted(    SectionInfo const& sectionInfo,
                                        Counts& assertions ) = 0;
        virtual void sectionEnded( SectionEndInfo const& endInfo ) = 0;
        virtual void sectionEndedEarly( SectionEndInfo const& endInfo ) = 0;

        virtual auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
) -> IGeneratorTracker& = 0;

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
        virtual void benchmarkPreparing( std::string const& name ) = 0;
        virtual void benchmarkStarting( BenchmarkInfo const& info ) = 0;
        virtual void benchmarkEnded( BenchmarkStats<> const& stats ) = 0;
        virtual void benchmarkFailed( std::string const& error ) = 0;
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

        virtual void pushScopedMessage( MessageInfo const& message ) = 0;
        virtual void popScopedMessage( MessageInfo const& message ) = 0;

        virtual void emplaceUnscopedMessage( MessageBuilder const& builder ) = 0;

        virtual void handleFatalErrorCondition( StringRef message ) = 0;

        virtual void handleExpr
                (   AssertionInfo const& info,
                        ITransientExpression const& expr,
                        AssertionReaction& reaction ) = 0;
        virtual void handleMessage
                (   AssertionInfo const& info,
                    ResultWas::OfType resultType,
                    StringRef const& message,
                    AssertionReaction& reaction ) = 0;
        virtual void handleUnexpectedExceptionNotThrown
                (   AssertionInfo const& info,
                    AssertionReaction& reaction ) = 0;
        virtual void handleUnexpectedInflightException
                (   AssertionInfo const& info,
                    std::string const& message,
                    AssertionReaction& reaction ) = 0;
        virtual void handleIncomplete
                (   AssertionInfo const& info ) = 0;
        virtual void handleNonExpr
                (   AssertionInfo const &info,
                    ResultWas::OfType resultType,
                    AssertionReaction &reaction ) = 0;

        virtual bool lastAssertionPassed() = 0;
        virtual void assertionPassed() = 0;

        // Deprecated, do not use:
        virtual std::string getCurrentTestName() const = 0;
        virtual const AssertionResult* getLastResult() const = 0;
        virtual void exceptionEarlyReported() = 0;
    };

    IResultCapture& getResultCapture();
}

// end catch_interfaces_capture.h
namespace Catch {

    struct TestFailureException{};
    struct AssertionResultData;
    struct IResultCapture;
    class RunContext;

    class LazyExpression {
        friend class AssertionHandler;
        friend struct AssertionStats;
        friend class RunContext;

        ITransientExpression const* m_transientExpression = nullptr;
        bool m_isNegated;
    public:
        LazyExpression( bool isNegated );
        LazyExpression( LazyExpression const& other );
        LazyExpression& operator = ( LazyExpression const& ) = delete;

        explicit operator bool() const;

        friend auto operator « ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream&;
    };

    struct AssertionReaction {
        bool shouldDebugBreak = false;
        bool shouldThrow = false;
    };

    class AssertionHandler {
        AssertionInfo m_assertionInfo;
        AssertionReaction m_reaction;
        bool m_completed = false;
        IResultCapture& m_resultCapture;

    public:
        AssertionHandler
            (   StringRef const& macroName,
                SourceLineInfo const& lineInfo,
                StringRef capturedExpression,
                ResultDisposition::Flags resultDisposition );
        ~AssertionHandler() {
            if ( !m_completed ) {
                m_resultCapture.handleIncomplete( m_assertionInfo );
            }
        }

        template<typename T>
        void handleExpr( ExprLhs<T> const& expr ) {
            handleExpr( expr.makeUnaryExpr() );
        }
        void handleExpr( ITransientExpression const& expr );

        void handleMessage(ResultWas::OfType resultType, StringRef const& message);

        void handleExceptionThrownAsExpected();
        void handleUnexpectedExceptionNotThrown();
        void handleExceptionNotThrownAsExpected();
        void handleThrowingCallSkipped();
        void handleUnexpectedInflightException();

        void complete();
        void setCompleted();

        // query
        auto allowThrows() const -> bool;
    };

    void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString );

} // namespace Catch

// end catch_assertionhandler.h
// start catch_message.h

#include <string>
#include <vector>

namespace Catch {

    struct MessageInfo {
        MessageInfo(    StringRef const& _macroName,
                        SourceLineInfo const& _lineInfo,
                        ResultWas::OfType _type );

        StringRef macroName;
        std::string message;
        SourceLineInfo lineInfo;
        ResultWas::OfType type;
        unsigned int sequence;

        bool operator == ( MessageInfo const& other ) const;
        bool operator < ( MessageInfo const& other ) const;
    private:
        static unsigned int globalCount;
    };

    struct MessageStream {

        template<typename T>
        MessageStream& operator << ( T const& value ) {
            m_stream << value;
            return *this;
        }

        ReusableStringStream m_stream;
    };

    struct MessageBuilder : MessageStream {
        MessageBuilder( StringRef const& macroName,
                        SourceLineInfo const& lineInfo,
                        ResultWas::OfType type );

        template<typename T>
        MessageBuilder& operator << ( T const& value ) {
            m_stream << value;
            return *this;
        }

        MessageInfo m_info;
    };

    class ScopedMessage {
    public:
        explicit ScopedMessage( MessageBuilder const& builder );
        ScopedMessage( ScopedMessage& duplicate ) = delete;
        ScopedMessage( ScopedMessage&& old );
        ~ScopedMessage();

        MessageInfo m_info;
        bool m_moved;
    };

    class Capturer {
        std::vector<MessageInfo> m_messages;
        IResultCapture& m_resultCapture = getResultCapture();
        size_t m_captured = 0;
    public:
        Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType resultType,
StringRef names );
        ~Capturer();

        void captureValue( size_t index, std::string const& value );

        template<typename T>
        void captureValues( size_t index, T const& value ) {
            captureValue( index, Catch::Detail::stringify( value ) );
        }

        template<typename T, typename... Ts>
        void captureValues( size_t index, T const& value, Ts const&... values ) {
            captureValue( index, Catch::Detail::stringify(value) );
            captureValues( index+1, values... );
        }
    };

} // end namespace Catch

// end catch_message.h
#if !defined(CATCH_CONFIG_DISABLE)

#if !defined(CATCH_CONFIG_DISABLE_STRINGIFICATION)
  #define CATCH_INTERNAL_STRINGIFY(...) #__VA_ARGS__
#else
  #define CATCH_INTERNAL_STRINGIFY(...) "Disabled by CATCH_CONFIG_DISABLE_STRINGIFICATION"
#endif

#if defined(CATCH_CONFIG_FAST_COMPILE) || defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)

// Another way to speed-up compilation is to omit local try-catch for REQUIRE*
// macros.
#define INTERNAL_CATCH_TRY
#define INTERNAL_CATCH_CATCH( capturer )

#else // CATCH_CONFIG_FAST_COMPILE

#define INTERNAL_CATCH_TRY try
#define INTERNAL_CATCH_CATCH( handler ) catch(...) { handler.handleUnexpectedInflightException(); }

#endif

#define INTERNAL_CATCH_REACT( handler ) handler.complete();

#define INTERNAL_CATCH_TEST( macroName, resultDisposition, ... ) \
    do { \
        CATCH_INTERNAL_IGNORE_BUT_WARN(__VA_ARGS__); \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
        INTERNAL_CATCH_TRY { \
            CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
            CATCH_INTERNAL_SUPPRESS_PARENTHESES_WARNINGS \
            catchAssertionHandler.handleExpr( Catch::Decomposer() <= __VA_ARGS__ ); \
            CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
        } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( (void)0, (false) && static_cast<bool>( !!(__VA_ARGS__) ) )

#define INTERNAL_CATCH_IF( macroName, resultDisposition, ... ) \
    INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
    if( Catch::getResultCapture().lastAssertionPassed() )

#define INTERNAL_CATCH_ELSE( macroName, resultDisposition, ... ) \
    INTERNAL_CATCH_TEST( macroName, resultDisposition, __VA_ARGS__ ); \
    if( !Catch::getResultCapture().lastAssertionPassed() )

#define INTERNAL_CATCH_NO_THROW( macroName, resultDisposition, ... ) \
    do { \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition ); \
        try { \
            static_cast<void>(__VA_ARGS__); \
            catchAssertionHandler.handleExceptionNotThrownAsExpected(); \
        } \
        catch( ... ) { \
            catchAssertionHandler.handleUnexpectedInflightException(); \
        } \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( false )

#define INTERNAL_CATCH_THROWS( macroName, resultDisposition, ... ) \
    do { \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__), resultDisposition); \
        if( catchAssertionHandler.allowThrows() ) \
            try { \
                static_cast<void>(__VA_ARGS__); \
                catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
            } \
            catch( ... ) { \
                catchAssertionHandler.handleExceptionThrownAsExpected(); \
            } \
        else \
            catchAssertionHandler.handleThrowingCallSkipped(); \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( false )

#define INTERNAL_CATCH_THROWS_AS( macroName, exceptionType, resultDisposition, expr ) \
    do { \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(expr) ", " CATCH_INTERNAL_STRINGIFY(exceptionType), resultDisposition ); \
        if( catchAssertionHandler.allowThrows() ) \
            try { \
                static_cast<void>(expr); \
                catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
            } \
            catch( exceptionType const& ) { \
                catchAssertionHandler.handleExceptionThrownAsExpected(); \
            } \
            catch( ... ) { \
                catchAssertionHandler.handleUnexpectedInflightException(); \
            } \
        else \
            catchAssertionHandler.handleThrowingCallSkipped(); \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( false )

#define INTERNAL_CATCH_MSG( macroName, messageType, resultDisposition, ... ) \
    do { \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
Catch::StringRef(), resultDisposition ); \
        catchAssertionHandler.handleMessage( messageType, ( Catch::MessageStream() « __VA_ARGS__ +
::Catch::StreamEndStop() ).m_stream.str() ); \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( false )

#define INTERNAL_CATCH_CAPTURE( varName, macroName, ... ) \
    auto varName = Catch::Capturer( macroName, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info,
#__VA_ARGS__ ); \
    varName.captureValues( 0, __VA_ARGS__ )

#define INTERNAL_CATCH_INFO( macroName, log ) \
    Catch::ScopedMessage INTERNAL_CATCH_UNIQUE_NAME( scopedMessage )( Catch::MessageBuilder(
macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) « log );

#define INTERNAL_CATCH_UNSCOPED_INFO( macroName, log ) \
    Catch::getResultCapture().emplaceUnscopedMessage( Catch::MessageBuilder( macroName##_catch_sr,
CATCH_INTERNAL_LINEINFO, Catch::ResultWas::Info ) « log )

// Although this is matcher-based, it can be used with just a string
#define INTERNAL_CATCH_THROWS_STR_MATCHES( macroName, resultDisposition, matcher, ... ) \
    do { \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO,
CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
        if( catchAssertionHandler.allowThrows() ) \
            try { \
                static_cast<void>(__VA_ARGS__); \
                catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
            } \
            catch( ... ) { \
                Catch::handleExceptionMatchExpr( catchAssertionHandler, matcher, #matcher##_catch_sr
); \
            } \
        else \
            catchAssertionHandler.handleThrowingCallSkipped(); \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( false )

#endif // CATCH_CONFIG_DISABLE

// end catch_capture.hpp
// start catch_section.h

// start catch_section_info.h

// start catch_totals.h

#include <cstddef>

namespace Catch {

    struct Counts {
        Counts operator - ( Counts const& other ) const;
        Counts& operator += ( Counts const& other );

        std::size_t total() const;
        bool allPassed() const;
        bool allOk() const;

        std::size_t passed = 0;
        std::size_t failed = 0;
        std::size_t failedButOk = 0;
    };

    struct Totals {

        Totals operator - ( Totals const& other ) const;
        Totals& operator += ( Totals const& other );

        Totals delta( Totals const& prevTotals ) const;

        int error = 0;
        Counts assertions;
        Counts testCases;
    };
}

// end catch_totals.h
#include <string>

namespace Catch {

    struct SectionInfo {
        SectionInfo
            (   SourceLineInfo const& _lineInfo,
                std::string const& _name );

        // Deprecated
        SectionInfo
            (   SourceLineInfo const& _lineInfo,
                std::string const& _name,
                std::string const& ) : SectionInfo( _lineInfo, _name ) {}

        std::string name;
        std::string description; // !Deprecated: this will always be empty
        SourceLineInfo lineInfo;
    };

    struct SectionEndInfo {
        SectionInfo sectionInfo;
        Counts prevAssertions;
        double durationInSeconds;
    };

} // end namespace Catch

// end catch_section_info.h
// start catch_timer.h

#include <cstdint>

namespace Catch {

    auto getCurrentNanosecondsSinceEpoch() -> uint64_t;
    auto getEstimatedClockResolution() -> uint64_t;

    class Timer {
        uint64_t m_nanoseconds = 0;
    public:
        void start();
        auto getElapsedNanoseconds() const -> uint64_t;
        auto getElapsedMicroseconds() const -> uint64_t;
        auto getElapsedMilliseconds() const -> unsigned int;
        auto getElapsedSeconds() const -> double;
    };

} // namespace Catch

// end catch_timer.h
#include <string>

namespace Catch {

    class Section : NonCopyable {
    public:
        Section( SectionInfo const& info );
        ~Section();

        // This indicates whether the section should be executed or not
        explicit operator bool() const;

    private:
        SectionInfo m_info;

        std::string m_name;
        Counts m_assertions;
        bool m_sectionIncluded;
        Timer m_timer;
    };

} // end namespace Catch

#define INTERNAL_CATCH_SECTION( ... ) \
    CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
    CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
    if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, __VA_ARGS__ ) ) \
    CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

#define INTERNAL_CATCH_DYNAMIC_SECTION( ... ) \
    CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
    CATCH_INTERNAL_SUPPRESS_UNUSED_WARNINGS \
    if( Catch::Section const& INTERNAL_CATCH_UNIQUE_NAME( catch_internal_Section ) = Catch::SectionInfo( CATCH_INTERNAL_LINEINFO, (Catch::ReusableStringStream() « __VA_ARGS__).str() ) ) \
    CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

// end catch_section.h
// start catch_interfaces_exception.h

// start catch_interfaces_registry_hub.h

#include <string>
#include <memory>

namespace Catch {

    class TestCase;
    struct ITestCaseRegistry;
    struct IExceptionTranslatorRegistry;
    struct IExceptionTranslator;
    struct IReporterRegistry;
    struct IReporterFactory;
    struct ITagAliasRegistry;
    struct IMutableEnumValuesRegistry;

    class StartupExceptionRegistry;

    using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;

    struct IRegistryHub {
        virtual ~IRegistryHub();

        virtual IReporterRegistry const& getReporterRegistry() const = 0;
        virtual ITestCaseRegistry const& getTestCaseRegistry() const = 0;
        virtual ITagAliasRegistry const& getTagAliasRegistry() const = 0;
        virtual IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const = 0;

        virtual StartupExceptionRegistry const& getStartupExceptionRegistry() const = 0;
    };

    struct IMutableRegistryHub {
        virtual ~IMutableRegistryHub();
        virtual void registerReporter( std::string const& name, IReporterFactoryPtr const& factory ) = 0;
        virtual void registerListener( IReporterFactoryPtr const& factory ) = 0;
        virtual void registerTest( TestCase const& testInfo ) = 0;
        virtual void registerTranslator( const IExceptionTranslator* translator ) = 0;
        virtual void registerTagAlias( std::string const& alias, std::string const& tag, SourceLineInfo const& lineInfo ) = 0;
        virtual void registerStartupException() noexcept = 0;
        virtual IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() = 0;
    };

    IRegistryHub const& getRegistryHub();
    IMutableRegistryHub& getMutableRegistryHub();
    void cleanUp();
    std::string translateActiveException();

}

// end catch_interfaces_registry_hub.h
#if defined(CATCH_CONFIG_DISABLE)
    #define INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG( translatorName, signature) \
        static std::string translatorName( signature )
#endif

#include <exception>
#include <string>
```

```
#include <vector>

namespace Catch {
    using exceptionTranslateFunction = std::string(*)();

    struct IExceptionTranslator;
    using ExceptionTranslators = std::vector<std::unique_ptr<IExceptionTranslator const»;

    struct IExceptionTranslator {
        virtual ~IExceptionTranslator();
        virtual std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const = 0;
    };

    struct IExceptionTranslatorRegistry {
        virtual ~IExceptionTranslatorRegistry();

        virtual std::string translateActiveException() const = 0;
    };

    class ExceptionTranslatorRegistrar {
        template<typename T>
        class ExceptionTranslator : public IExceptionTranslator {
        public:

            ExceptionTranslator( std::string(*translateFunction)( T& ) )
            : m_translateFunction( translateFunction )
            {}

            std::string translate( ExceptionTranslators::const_iterator it,
ExceptionTranslators::const_iterator itEnd ) const override {
#if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
                return "";
#else
                try {
                    if( it == itEnd )
                        std::rethrow_exception(std::current_exception());
                    else
                        return (*it)->translate( it+1, itEnd );
                }
                catch( T& ex ) {
                    return m_translateFunction( ex );
                }
#endif
            }

        protected:
            std::string(*m_translateFunction)( T& );
        };

    public:
        template<typename T>
        ExceptionTranslatorRegistrar( std::string(*translateFunction)( T& ) ) {
            getMutableRegistryHub().registerTranslator
                ( new ExceptionTranslator<T>( translateFunction ) );
        }
    };
}

#define INTERNAL_CATCH_TRANSLATE_EXCEPTION2( translatorName, signature ) \
    static std::string translatorName( signature ); \
    CATCH_INTERNAL_START_WARNINGS_SUPPRESSION \
    CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS \
    namespace{ Catch::ExceptionTranslatorRegistrar INTERNAL_CATCH_UNIQUE_NAME(
catch_internal_ExceptionRegistrar )( &translatorName ); } \
    CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION \
    static std::string translatorName( signature )

#define INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION2(
INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )

// end catch_interfaces_exception.h
// start catch_approx.h

#include <type_traits>

namespace Catch {
namespace Detail {

    class Approx {
    private:
        bool equalityComparisonImpl(double other) const;
        // Validates the new margin (margin >= 0)
        // out-of-line to avoid including stdexcept in the header
        void setMargin(double margin);
        // Validates the new epsilon (0 < epsilon < 1)
        // out-of-line to avoid including stdexcept in the header
        void setEpsilon(double epsilon);

    public:
        explicit Approx ( double value );

        static Approx custom();

        Approx operator-() const;

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        Approx operator()( T const& value ) const {
            Approx approx( static_cast<double>(value) );
            approx.m_epsilon = m_epsilon;
            approx.m_margin = m_margin;
            approx.m_scale = m_scale;
            return approx;
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        explicit Approx( T const& value ): Approx(static_cast<double>(value))
        {}

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator == ( const T& lhs, Approx const& rhs ) {
            auto lhs_v = static_cast<double>(lhs);
            return rhs.equalityComparisonImpl(lhs_v);
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator == ( Approx const& lhs, const T& rhs ) {
            return operator==( rhs, lhs );
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator != ( T const& lhs, Approx const& rhs ) {
            return !operator==( lhs, rhs );
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator != ( Approx const& lhs, T const& rhs ) {
            return !operator==( rhs, lhs );
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator <= ( T const& lhs, Approx const& rhs ) {
            return static_cast<double>(lhs) < rhs.m_value || lhs == rhs;
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator <= ( Approx const& lhs, T const& rhs ) {
            return lhs.m_value < static_cast<double>(rhs) || lhs == rhs;
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator >= ( T const& lhs, Approx const& rhs ) {
            return static_cast<double>(lhs) > rhs.m_value || lhs == rhs;
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        friend bool operator >= ( Approx const& lhs, T const& rhs ) {
            return lhs.m_value > static_cast<double>(rhs) || lhs == rhs;
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        Approx& epsilon( T const& newEpsilon ) {
            double epsilonAsDouble = static_cast<double>(newEpsilon);
            setEpsilon(epsilonAsDouble);
            return *this;
        }

template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        Approx& margin( T const& newMargin ) {
            double marginAsDouble = static_cast<double>(newMargin);
            setMargin(marginAsDouble);
            return *this;
        }


template <typename T, typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
        Approx& scale( T const& newScale ) {
            m_scale = static_cast<double>(newScale);
            return *this;
        }

        std::string toString() const;

    private:
        double m_epsilon;
        double m_margin;
        double m_scale;
        double m_value;
    };
} // end namespace Detail

namespace literals {
    Detail::Approx operator "" _a(long double val);
    Detail::Approx operator "" _a(unsigned long long val);
} // end namespace literals

template<>
struct StringMaker<Catch::Detail::Approx> {
    static std::string convert(Catch::Detail::Approx const& value);
};

} // end namespace Catch

// end catch_approx.h
// start catch_string_manip.h

#include <string>
#include <iosfwd>
#include <vector>

namespace Catch {

    bool startsWith( std::string const& s, std::string const& prefix );
    bool startsWith( std::string const& s, char prefix );
    bool endsWith( std::string const& s, std::string const& suffix );
    bool endsWith( std::string const& s, char suffix );
    bool contains( std::string const& s, std::string const& infix );
    void toLowerInPlace( std::string& s );
    std::string toLower( std::string const& s );
    std::string trim( std::string const& str );
    StringRef trim(StringRef ref);

    // !!! Be aware, returns refs into original string - make sure original string outlives them
    std::vector<StringRef> splitStringRef( StringRef str, char delimiter );
    bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis
);

    struct pluralise {
        pluralise( std::size_t count, std::string const& label );

        friend std::ostream& operator « ( std::ostream& os, pluralise const& pluraliser );

        std::size_t m_count;
        std::string m_label;
    };
}

// end catch_string_manip.h
#ifndef CATCH_CONFIG_DISABLE_MATCHERS
// start catch_capture_matchers.h

// start catch_matchers.h

#include <string>
#include <vector>

namespace Catch {
namespace Matchers {
    namespace Impl {

        template<typename ArgT> struct MatchAllOf;
        template<typename ArgT> struct MatchAnyOf;
        template<typename ArgT> struct MatchNotOf;

        class MatcherUntypedBase {
        public:
            MatcherUntypedBase() = default;
            MatcherUntypedBase ( MatcherUntypedBase const& ) = default;
            MatcherUntypedBase& operator = ( MatcherUntypedBase const& ) = delete;
            std::string toString() const;

        protected:
            virtual ~MatcherUntypedBase();
            virtual std::string describe() const = 0;
            mutable std::string m_cachedToString;
        };

#ifdef __clang__
#    pragma clang diagnostic push
#    pragma clang diagnostic ignored "-Wnon-virtual-dtor"
#endif

        template<typename ObjectT>
        struct MatcherMethod {
            virtual bool match( ObjectT const& arg ) const = 0;
        };

#if defined(__OBJC__)
        // Hack to fix Catch GH issue #1661. Could use id for generic Object support.
        // use of const for Object pointers is very uncommon and under ARC it causes some kind of
signature mismatch that breaks compilation
        template<>
        struct MatcherMethod<NSString*> {
            virtual bool match( NSString* arg ) const = 0;
        };
#endif

#ifdef __clang__
#    pragma clang diagnostic pop
#endif

        template<typename T>
        struct MatcherBase : MatcherUntypedBase, MatcherMethod<T> {

            MatchAllOf<T> operator && ( MatcherBase const& other ) const;
            MatchAnyOf<T> operator || ( MatcherBase const& other ) const;
            MatchNotOf<T> operator ! () const;
        };

        template<typename ArgT>
        struct MatchAllOf : MatcherBase<ArgT> {
            bool match( ArgT const& arg ) const override {
                for( auto matcher : m_matchers ) {
                    if (!matcher->match(arg))
                        return false;
                }
                return true;
            }
            std::string describe() const override {
                std::string description;
                description.reserve( 4 + m_matchers.size()*32 );
                description += "( ";
                bool first = true;
                for( auto matcher : m_matchers ) {
                    if( first )
                        first = false;
                    else
                        description += " and ";
                    description += matcher->toString();
                }
                description += " )";
                return description;
            }

            MatchAllOf<ArgT> operator && ( MatcherBase<ArgT> const& other ) {
                auto copy(*this);
                copy.m_matchers.push_back( &other );
                return copy;
            }

            std::vector<MatcherBase<ArgT> const*> m_matchers;
        };
        template<typename ArgT>
        struct MatchAnyOf : MatcherBase<ArgT> {

            bool match( ArgT const& arg ) const override {
                for( auto matcher : m_matchers ) {
                    if (matcher->match(arg))
                        return true;
                }
                return false;
            }
            std::string describe() const override {
                std::string description;
                description.reserve( 4 + m_matchers.size()*32 );
                description += "( ";
                bool first = true;
                for( auto matcher : m_matchers ) {
                    if( first )
                        first = false;
                    else
                        description += " or ";
                    description += matcher->toString();
                }
                description += " )";
                return description;
            }

            MatchAnyOf<ArgT> operator || ( MatcherBase<ArgT> const& other ) {
                auto copy(*this);
                copy.m_matchers.push_back( &other );
                return copy;
            }

            std::vector<MatcherBase<ArgT> const*> m_matchers;
        };

        template<typename ArgT>
        struct MatchNotOf : MatcherBase<ArgT> {

            MatchNotOf( MatcherBase<ArgT> const& underlyingMatcher ) : m_underlyingMatcher(
underlyingMatcher ) {}

            bool match( ArgT const& arg ) const override {
                return !m_underlyingMatcher.match( arg );
            }

            std::string describe() const override {
                return "not " + m_underlyingMatcher.toString();
            }
            MatcherBase<ArgT> const& m_underlyingMatcher;
        };

        template<typename T>
        MatchAllOf<T> MatcherBase<T>::operator && ( MatcherBase const& other ) const {
            return MatchAllOf<T>() && *this && other;
        }
        template<typename T>
        MatchAnyOf<T> MatcherBase<T>::operator || ( MatcherBase const& other ) const {
            return MatchAnyOf<T>() || *this || other;
        }
        template<typename T>
        MatchNotOf<T> MatcherBase<T>::operator ! () const {
            return MatchNotOf<T>( *this );
        }

    } // namespace Impl

} // namespace Matchers

using namespace Matchers;
using Matchers::Impl::MatcherBase;

} // namespace Catch

// end catch_matchers.h
// start catch_matchers_exception.hpp

namespace Catch {
namespace Matchers {
namespace Exception {

class ExceptionMessageMatcher : public MatcherBase<std::exception> {
    std::string m_message;
public:

    ExceptionMessageMatcher(std::string const& message):
        m_message(message)
    {}

    bool match(std::exception const& ex) const override;

    std::string describe() const override;
};

} // namespace Exception

Exception::ExceptionMessageMatcher Message(std::string const& message);

} // namespace Matchers
} // namespace Catch

// end catch_matchers_exception.hpp
// start catch_matchers_floating.h

namespace Catch {
namespace Matchers {

    namespace Floating {

        enum class FloatingPointKind : uint8_t;

        struct WithinAbsMatcher : MatcherBase<double> {
            WithinAbsMatcher(double target, double margin);
            bool match(double const& matchee) const override;
            std::string describe() const override;
        private:
            double m_target;
            double m_margin;
        };

        struct WithinUlpsMatcher : MatcherBase<double> {
            WithinUlpsMatcher(double target, uint64_t ulps, FloatingPointKind baseType);
            bool match(double const& matchee) const override;
            std::string describe() const override;
        private:
            double m_target;
            uint64_t m_ulps;
            FloatingPointKind m_type;
        };

        // Given IEEE-754 format for floats and doubles, we can assume
        // that float -> double promotion is lossless. Given this, we can
        // assume that if we do the standard relative comparison of
        // |lhs - rhs| <= epsilon * max(fabs(lhs), fabs(rhs)), then we get
        // the same result if we do this for floats, as if we do this for
        // doubles that were promoted from floats.
        struct WithinRelMatcher : MatcherBase<double> {
            WithinRelMatcher(double target, double epsilon);
            bool match(double const& matchee) const override;
            std::string describe() const override;
        private:
            double m_target;
            double m_epsilon;
        };

    } // namespace Floating

    // The following functions create the actual matcher objects.
    // This allows the types to be inferred
    Floating::WithinUlpsMatcher WithinULP(double target, uint64_t maxUlpDiff);
    Floating::WithinUlpsMatcher WithinULP(float target, uint64_t maxUlpDiff);
    Floating::WithinAbsMatcher WithinAbs(double target, double margin);
    Floating::WithinRelMatcher WithinRel(double target, double eps);
    // defaults epsilon to 100*numeric_limits<double>::epsilon()
    Floating::WithinRelMatcher WithinRel(double target);
    Floating::WithinRelMatcher WithinRel(float target, float eps);
    // defaults epsilon to 100*numeric_limits<float>::epsilon()
    Floating::WithinRelMatcher WithinRel(float target);

} // namespace Matchers
} // namespace Catch

// end catch_matchers_floating.h
// start catch_matchers_generic.hpp

#include <functional>
#include <string>

namespace Catch {
namespace Matchers {
namespace Generic {

namespace Detail {
    std::string finalizeDescription(const std::string& desc);
}

template <typename T>
class PredicateMatcher : public MatcherBase<T> {
    std::function<bool(T const&)> m_predicate;
    std::string m_description;
public:

    PredicateMatcher(std::function<bool(T const&)> const& elem, std::string const& descr)
        :m_predicate(std::move(elem)),
        m_description(Detail::finalizeDescription(descr))
    {}

    bool match( T const& item ) const override {
        return m_predicate(item);
    }

    std::string describe() const override {
        return m_description;
    }
};

} // namespace Generic

    // The following functions create the actual matcher objects.
    // The user has to explicitly specify type to the function, because
    // inferring std::function<bool(T const&)> is hard (but possible) and
    // requires a lot of TMP.
    template<typename T>
    Generic::PredicateMatcher<T> Predicate(std::function<bool(T const&)> const& predicate, std::string
const& description = "") {
        return Generic::PredicateMatcher<T>(predicate, description);
    }

} // namespace Matchers
} // namespace Catch

// end catch_matchers_generic.hpp
// start catch_matchers_string.h

#include <string>

namespace Catch {
namespace Matchers {

    namespace StdString {

        struct CasedString
        {
            CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity );
            std::string adjustString( std::string const& str ) const;
            std::string caseSensitivitySuffix() const;

            CaseSensitive::Choice m_caseSensitivity;
            std::string m_str;
        };

        struct StringMatcherBase : MatcherBase<std::string> {
            StringMatcherBase( std::string const& operation, CasedString const& comparator );
            std::string describe() const override;

            CasedString m_comparator;
            std::string m_operation;
        };

        struct EqualsMatcher : StringMatcherBase {
            EqualsMatcher( CasedString const& comparator );
            bool match( std::string const& source ) const override;
        };
        struct ContainsMatcher : StringMatcherBase {
            ContainsMatcher( CasedString const& comparator );
            bool match( std::string const& source ) const override;
        };
        struct StartsWithMatcher : StringMatcherBase {
            StartsWithMatcher( CasedString const& comparator );
            bool match( std::string const& source ) const override;
        };
        struct EndsWithMatcher : StringMatcherBase {
            EndsWithMatcher( CasedString const& comparator );
            bool match( std::string const& source ) const override;
        };

        struct RegexMatcher : MatcherBase<std::string> {
            RegexMatcher( std::string regex, CaseSensitive::Choice caseSensitivity );
            bool match( std::string const& matchee ) const override;
            std::string describe() const override;

        private:
            std::string m_regex;
            CaseSensitive::Choice m_caseSensitivity;
        };

    } // namespace StdString

    // The following functions create the actual matcher objects.
    // This allows the types to be inferred

    StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );
    StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
    StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity
= CaseSensitive::Yes );
    StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity = CaseSensitive::Yes );
    StdString::RegexMatcher Matches( std::string const& regex, CaseSensitive::Choice caseSensitivity =
CaseSensitive::Yes );

} // namespace Matchers
} // namespace Catch

// end catch_matchers_string.h
// start catch_matchers_vector.h

#include <algorithm>

namespace Catch {
namespace Matchers {

    namespace Vector {
        template<typename T, typename Alloc>
        struct ContainsElementMatcher : MatcherBase<std::vector<T, Alloc» {

            ContainsElementMatcher(T const &comparator) : m_comparator( comparator) {}

            bool match(std::vector<T, Alloc> const &v) const override {
                for (auto const& el : v) {
                    if (el == m_comparator) {
                        return true;
                    }
                }
                return false;
            }

            std::string describe() const override {
                return "Contains: " + ::Catch::Detail::stringify( m_comparator );
            }

            T const& m_comparator;
        };

        template<typename T, typename AllocComp, typename AllocMatch>
        struct ContainsMatcher : MatcherBase<std::vector<T, AllocMatch» {

            ContainsMatcher(std::vector<T, AllocComp> const &comparator) : m_comparator( comparator )
{}

            bool match(std::vector<T, AllocMatch> const &v) const override {
                // !TBD: see note in EqualsMatcher
                if (m_comparator.size() > v.size())
                    return false;
                for (auto const& comparator : m_comparator) {
                    auto present = false;
                    for (const auto& el : v) {
                        if (el == comparator) {
                            present = true;
                            break;
                        }
                    }
                    if (!present) {
                        return false;
                    }
                }
                return true;
            }
            std::string describe() const override {
                return "Contains: " + ::Catch::Detail::stringify( m_comparator );
            }

            std::vector<T, AllocComp> const& m_comparator;
        };

        template<typename T, typename AllocComp, typename AllocMatch>
        struct EqualsMatcher : MatcherBase<std::vector<T, AllocMatch» {

            EqualsMatcher(std::vector<T, AllocComp> const &comparator) : m_comparator( comparator ) {}

            bool match(std::vector<T, AllocMatch> const &v) const override {
                // !TBD: This currently works if all elements can be compared using !=
                // - a more general approach would be via a compare template that defaults
                // to using !=. but could be specialised for, e.g. std::vector<T, Alloc> etc
                // - then just call that directly
                if (m_comparator.size() != v.size())
                    return false;
                for (std::size_t i = 0; i < v.size(); ++i)
                    if (m_comparator[i] != v[i])
                        return false;
                return true;
            }
```

```
      std::string describe() const override {
          return "Equals: " + ::Catch::Detail::stringify( m_comparator );
      }
      std::vector<T, AllocComp> const& m_comparator;
    };

    template<typename T, typename AllocComp, typename AllocMatch>
    struct ApproxMatcher : MatcherBase<std::vector<T, AllocMatch» {

      ApproxMatcher(std::vector<T, AllocComp> const& comparator) : m_comparator( comparator ) {}

      bool match(std::vector<T, AllocMatch> const &v) const override {
          if (m_comparator.size() != v.size())
              return false;
          for (std::size_t i = 0; i < v.size(); ++i)
              if (m_comparator[i] != approx(v[i]))
                  return false;
          return true;
      }
      std::string describe() const override {
          return "is approx: " + ::Catch::Detail::stringify( m_comparator );
      }

      template <typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
      ApproxMatcher& epsilon( T const& newEpsilon ) {
          approx.epsilon(newEpsilon);
          return *this;
      }

      template <typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
      ApproxMatcher& margin( T const& newMargin ) {
          approx.margin(newMargin);
          return *this;
      }

      template <typename = typename std::enable_if<std::is_constructible<double, T>::value>::type>
      ApproxMatcher& scale( T const& newScale ) {
          approx.scale(newScale);
          return *this;
      }

      std::vector<T, AllocComp> const& m_comparator;
      mutable Catch::Detail::Approx approx = Catch::Detail::Approx::custom();
    };

    template<typename T, typename AllocComp, typename AllocMatch>
    struct UnorderedEqualsMatcher : MatcherBase<std::vector<T, AllocMatch» {
      UnorderedEqualsMatcher(std::vector<T, AllocComp> const& target) : m_target(target) {}
      bool match(std::vector<T, AllocMatch> const& vec) const override {
          if (m_target.size() != vec.size()) {
              return false;
          }
          return std::is_permutation(m_target.begin(), m_target.end(), vec.begin());
      }

      std::string describe() const override {
          return "UnorderedEquals: " + ::Catch::Detail::stringify(m_target);
      }
    private:
      std::vector<T, AllocComp> const& m_target;
    };

  } // namespace Vector

  // The following functions create the actual matcher objects.
  // This allows the types to be inferred

  template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
  Vector::ContainsMatcher<T, AllocComp, AllocMatch> Contains( std::vector<T, AllocComp> const& comparator ) {
    return Vector::ContainsMatcher<T, AllocComp, AllocMatch>( comparator );
  }

  template<typename T, typename Alloc = std::allocator<T»
  Vector::ContainsElementMatcher<T, Alloc> VectorContains( T const& comparator ) {
    return Vector::ContainsElementMatcher<T, Alloc>( comparator );
  }

  template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
  Vector::EqualsMatcher<T, AllocComp, AllocMatch> Equals( std::vector<T, AllocComp> const& comparator ) {
    return Vector::EqualsMatcher<T, AllocComp, AllocMatch>( comparator );
  }

  template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
  Vector::ApproxMatcher<T, AllocComp, AllocMatch> Approx( std::vector<T, AllocComp> const& comparator ) {
        return Vector::ApproxMatcher<T, AllocComp, AllocMatch>( comparator );
    }

    template<typename T, typename AllocComp = std::allocator<T>, typename AllocMatch = AllocComp>
    Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch> UnorderedEquals(std::vector<T, AllocComp> const& target) {
        return Vector::UnorderedEqualsMatcher<T, AllocComp, AllocMatch>( target );
    }

} // namespace Matchers
} // namespace Catch

// end catch_matchers_vector.h
namespace Catch {

    template<typename ArgT, typename MatcherT>
    class MatchExpr : public ITransientExpression {
        ArgT const& m_arg;
        MatcherT m_matcher;
        StringRef m_matcherString;
    public:
        MatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString )
        :   ITransientExpression{ true, matcher.match( arg ) },
            m_arg( arg ),
            m_matcher( matcher ),
            m_matcherString( matcherString )
        {}

        void streamReconstructedExpression( std::ostream &os ) const override {
            auto matcherAsString = m_matcher.toString();
            os « Catch::Detail::stringify( m_arg ) « ' ';
            if( matcherAsString == Detail::unprintableString )
                os « m_matcherString;
            else
                os « matcherAsString;
        }
    };

    using StringMatcher = Matchers::Impl::MatcherBase<std::string>;

    void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef const& matcherString  );

    template<typename ArgT, typename MatcherT>
    auto makeMatchExpr( ArgT const& arg, MatcherT const& matcher, StringRef const& matcherString  ) -> MatchExpr<ArgT, MatcherT> {
        return MatchExpr<ArgT, MatcherT>( arg, matcher, matcherString );
    }

} // namespace Catch

#define INTERNAL_CHECK_THAT( macroName, matcher, resultDisposition, arg ) \
    do { \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, CATCH_INTERNAL_STRINGIFY(arg) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
        INTERNAL_CATCH_TRY { \
            catchAssertionHandler.handleExpr( Catch::makeMatchExpr( arg, matcher, #matcher##_catch_sr ) ); \
        } INTERNAL_CATCH_CATCH( catchAssertionHandler ) \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( false )

#define INTERNAL_CATCH_THROWS_MATCHES( macroName, exceptionType, resultDisposition, matcher, ... ) \
    do { \
        Catch::AssertionHandler catchAssertionHandler( macroName##_catch_sr, CATCH_INTERNAL_LINEINFO, CATCH_INTERNAL_STRINGIFY(__VA_ARGS__) ", " CATCH_INTERNAL_STRINGIFY(exceptionType) ", " CATCH_INTERNAL_STRINGIFY(matcher), resultDisposition ); \
        if( catchAssertionHandler.allowThrows() ) \
            try { \
                static_cast<void>(__VA_ARGS__ ); \
                catchAssertionHandler.handleUnexpectedExceptionNotThrown(); \
            } \
            catch( exceptionType const& ex ) { \
                catchAssertionHandler.handleExpr( Catch::makeMatchExpr( ex, matcher, #matcher##_catch_sr ) ); \
            } \
            catch( ... ) { \
                catchAssertionHandler.handleUnexpectedInflightException(); \
            } \
        else \
            catchAssertionHandler.handleThrowingCallSkipped(); \
        INTERNAL_CATCH_REACT( catchAssertionHandler ) \
    } while( false )

// end catch_capture_matchers.h
#endif
// start catch_generators.hpp

// start catch_interfaces_generatortracker.h


#include <memory>

namespace Catch {

    namespace Generators {
        class GeneratorUntypedBase {
        public:
            GeneratorUntypedBase() = default;
            virtual ~GeneratorUntypedBase();
            // Attempts to move the generator to the next element
             //
             // Returns true iff the move succeeded (and a valid element
             // can be retrieved).
            virtual bool next() = 0;
        };
        using GeneratorBasePtr = std::unique_ptr<GeneratorUntypedBase>;

    } // namespace Generators

    struct IGeneratorTracker {
        virtual ~IGeneratorTracker();
        virtual auto hasGenerator() const -> bool = 0;
        virtual auto getGenerator() const -> Generators::GeneratorBasePtr const& = 0;
        virtual void setGenerator( Generators::GeneratorBasePtr&& generator ) = 0;
    };

} // namespace Catch

// end catch_interfaces_generatortracker.h
// start catch_enforce.h

#include <exception>

namespace Catch {
#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
    template <typename Ex>
    [[noreturn]]
    void throw_exception(Ex const& e) {
        throw e;
    }
#else // ^^ Exceptions are enabled //  Exceptions are disabled vv
    [[noreturn]]
    void throw_exception(std::exception const& e);
#endif

    [[noreturn]]
    void throw_logic_error(std::string const& msg);
    [[noreturn]]
    void throw_domain_error(std::string const& msg);
    [[noreturn]]
    void throw_runtime_error(std::string const& msg);

} // namespace Catch;

#define CATCH_MAKE_MSG(...) \
    (Catch::ReusableStringStream() « __VA_ARGS__).str()

#define CATCH_INTERNAL_ERROR(...) \
    Catch::throw_logic_error(CATCH_MAKE_MSG( CATCH_INTERNAL_LINEINFO « ": Internal Catch2 error: " «
__VA_ARGS__))

#define CATCH_ERROR(...) \
    Catch::throw_domain_error(CATCH_MAKE_MSG( __VA_ARGS__ ))

#define CATCH_RUNTIME_ERROR(...) \
    Catch::throw_runtime_error(CATCH_MAKE_MSG( __VA_ARGS__ ))

#define CATCH_ENFORCE( condition, ... ) \
    do{ if( !(condition) ) CATCH_ERROR( __VA_ARGS__ ); } while(false)

// end catch_enforce.h
#include <memory>
#include <vector>
#include <cassert>

#include <utility>
#include <exception>

namespace Catch {

class GeneratorException : public std::exception {
    const char* const m_msg = "";

```

```
public:
    GeneratorException(const char* msg):
        m_msg(msg)
    {}

    const char* what() const noexcept override final;
};

namespace Generators {

    // !TBD move this into its own location?
    namespace pf{
        template<typename T, typename... Args>
        std::unique_ptr<T> make_unique( Args&&... args ) {
            return std::unique_ptr<T>(new T(std::forward<Args>(args)...));
        }
    }

    template<typename T>
    struct IGenerator : GeneratorUntypedBase {
        virtual ~IGenerator() = default;

        // Returns the current element of the generator
        //
        // \Precondition The generator is either freshly constructed,
        // or the last call to `next()` returned true
        virtual T const& get() const = 0;
        using type = T;
    };

    template<typename T>
    class SingleValueGenerator final : public IGenerator<T> {
        T m_value;
    public:
        SingleValueGenerator(T&& value) : m_value(std::move(value)) {}

        T const& get() const override {
            return m_value;
        }
        bool next() override {
            return false;
        }
    };

    template<typename T>
    class FixedValuesGenerator final : public IGenerator<T> {
        static_assert(!std::is_same<T, bool>::value,
            "FixedValuesGenerator does not support bools because of std::vector<bool>"
            "specialization, use SingleValue Generator instead.");
        std::vector<T> m_values;
        size_t m_idx = 0;
    public:
        FixedValuesGenerator( std::initializer_list<T> values ) : m_values( values ) {}

        T const& get() const override {
            return m_values[m_idx];
        }
        bool next() override {
            ++m_idx;
            return m_idx < m_values.size();
        }
    };

    template <typename T>
    class GeneratorWrapper final {
        std::unique_ptr<IGenerator<T» m_generator;
    public:
        GeneratorWrapper(std::unique_ptr<IGenerator<T>> generator):
            m_generator(std::move(generator))
        {}
        T const& get() const {
            return m_generator->get();
        }
        bool next() {
            return m_generator->next();
        }
    };

    template <typename T>
    GeneratorWrapper<T> value(T&& value) {
        return GeneratorWrapper<T>(pf::make_unique<SingleValueGenerator<T>>(std::forward<T>(value)));
    }
    template <typename T>
    GeneratorWrapper<T> values(std::initializer_list<T> values) {
        return GeneratorWrapper<T>(pf::make_unique<FixedValuesGenerator<T>>(values));
    }

    template<typename T>
    class Generators : public IGenerator<T> {
        std::vector<GeneratorWrapper<T>» m_generators;
        size_t m_current = 0;

        void populate(GeneratorWrapper<T>&& generator) {
            m_generators.emplace_back(std::move(generator));
        }
        void populate(T&& val) {
            m_generators.emplace_back(value(std::forward<T>(val)));
        }
        template<typename U>
        void populate(U&& val) {
            populate(T(std::forward<U>(val)));
        }
        template<typename U, typename... Gs>
        void populate(U&& valueOrGenerator, Gs &&... moreGenerators) {
            populate(std::forward<U>(valueOrGenerator));
            populate(std::forward<Gs>(moreGenerators)...);
        }

    public:
        template <typename... Gs>
        Generators(Gs &&... moreGenerators) {
            m_generators.reserve(sizeof...(Gs));
            populate(std::forward<Gs>(moreGenerators)...);
        }

        T const& get() const override {
            return m_generators[m_current].get();
        }

        bool next() override {
            if (m_current >= m_generators.size()) {
                return false;
            }
            const bool current_status = m_generators[m_current].next();
            if (!current_status) {
                ++m_current;
            }
            return m_current < m_generators.size();
        }
    };

    template<typename... Ts>
    GeneratorWrapper<std::tuple<Ts...» table( std::initializer_list<std::tuple<typename
std::decay<Ts>::type...» tuples ) {
        return values<std::tuple<Ts...»( tuples );
    }

    // Tag type to signal that a generator sequence should convert arguments to a specific type
    template <typename T>
    struct as {};

    template<typename T, typename... Gs>
    auto makeGenerators( GeneratorWrapper<T>&& generator, Gs &&... moreGenerators ) -> Generators<T> {
        return Generators<T>(std::move(generator), std::forward<Gs>(moreGenerators)...);
    }
    template<typename T>
    auto makeGenerators( GeneratorWrapper<T>&& generator ) -> Generators<T> {
        return Generators<T>(std::move(generator));
    }
    template<typename T, typename... Gs>
    auto makeGenerators( T&& val, Gs &&... moreGenerators ) -> Generators<T> {
        return makeGenerators( value( std::forward<T>( val ) ), std::forward<Gs>( moreGenerators )...
);
    }
    template<typename T, typename U, typename... Gs>
    auto makeGenerators( as<T>, U&& val, Gs &&... moreGenerators ) -> Generators<T> {
        return makeGenerators( value( T( std::forward<U>( val ) ) ), std::forward<Gs>( moreGenerators
)... );
    }

    auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
IGeneratorTracker&;

    template<typename L>
    // Note: The type after -> is weird, because VS2015 cannot parse
    //       the expression used in the typedef inside, when it is in
    //       return type. Yeah.
    auto generate( StringRef generatorName, SourceLineInfo const& lineInfo, L const&
generatorExpression ) -> decltype(std::declval<decltype(generatorExpression())>().get()) {
        using UnderlyingType = typename decltype(generatorExpression())::type;

        IGeneratorTracker& tracker = acquireGeneratorTracker( generatorName, lineInfo );
        if (!tracker.hasGenerator()) {
            tracker.setGenerator(pf::make_unique<Generators<UnderlyingType»(generatorExpression()));
        }

        auto const& generator = static_cast<IGenerator<UnderlyingType> const&>(
*tracker.getGenerator() );
        return generator.get();
    }

} // namespace Generators
} // namespace Catch

#define GENERATE( ... ) \
    Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
                                 CATCH_INTERNAL_LINEINFO, \
                                 [ ]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
#define GENERATE_COPY( ... ) \
    Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
                                 CATCH_INTERNAL_LINEINFO, \
                                 [=]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)
#define GENERATE_REF( ... ) \
    Catch::Generators::generate( INTERNAL_CATCH_STRINGIZE(INTERNAL_CATCH_UNIQUE_NAME(generator)), \
                                 CATCH_INTERNAL_LINEINFO, \
                                 [&]{ using namespace Catch::Generators; return makeGenerators(
__VA_ARGS__ ); } ) //NOLINT(google-build-using-namespace)

// end catch_generators.hpp
// start catch_generators_generic.hpp

namespace Catch {
namespace Generators {

    template <typename T>
    class TakeGenerator : public IGenerator<T> {
        GeneratorWrapper<T> m_generator;
        size_t m_returned = 0;
        size_t m_target;
    public:
        TakeGenerator(size_t target, GeneratorWrapper<T>&& generator):
            m_generator(std::move(generator)),
            m_target(target)
        {
            assert(target != 0 && "Empty generators are not allowed");
        }
        T const& get() const override {
            return m_generator.get();
        }
        bool next() override {
            ++m_returned;
            if (m_returned >= m_target) {
                return false;
            }

            const auto success = m_generator.next();
            // If the underlying generator does not contain enough values
            // then we cut short as well
            if (!success) {
                m_returned = m_target;
            }
            return success;
        }
    };

    template <typename T>
    GeneratorWrapper<T> take(size_t target, GeneratorWrapper<T>&& generator) {
        return GeneratorWrapper<T>(pf::make_unique<TakeGenerator<T>>(target, std::move(generator)));
    }

    template <typename T, typename Predicate>
    class FilterGenerator : public IGenerator<T> {
        GeneratorWrapper<T> m_generator;
        Predicate m_predicate;
    public:
        template <typename P = Predicate>
        FilterGenerator(P&& pred, GeneratorWrapper<T>&& generator):
            m_generator(std::move(generator)),
            m_predicate(std::forward<P>(pred))
        {
            if (!m_predicate(m_generator.get())) {
                // It might happen that there are no values that pass the
                // filter. In that case we throw an exception.
                auto has_initial_value = nextImpl();
                if (!has_initial_value) {
                    Catch::throw_exception(GeneratorException("No valid value found in filtered
generator"));
                }
            }
        }

        T const& get() const override {
            return m_generator.get();
        }

        bool next() override {
            return nextImpl();
        }

    private:
        bool nextImpl() {
            bool success = m_generator.next();
            if (!success) {
                return false;
            }
            while (!m_predicate(m_generator.get()) && (success = m_generator.next()) == true);
            return success;
        }
    };

    template <typename T, typename Predicate>
    GeneratorWrapper<T> filter(Predicate&& pred, GeneratorWrapper<T>&& generator) {
        return
GeneratorWrapper<T>(std::unique_ptr<IGenerator<T>>(pf::make_unique<FilterGenerator<T, Predicate>>(std::forward<Predicat
std::move(generator))));
    }

    template <typename T>
    class RepeatGenerator : public IGenerator<T> {
        static_assert(!std::is_same<T, bool>::value,
            "RepeatGenerator currently does not support bools"
            "because of std::vector<bool> specialization");
        GeneratorWrapper<T> m_generator;
        mutable std::vector<T> m_returned;
        size_t m_target_repeats;
        size_t m_current_repeat = 0;
        size_t m_repeat_index = 0;
    public:
        RepeatGenerator(size_t repeats, GeneratorWrapper<T>&& generator):
            m_generator(std::move(generator)),
            m_target_repeats(repeats)
        {
            assert(m_target_repeats > 0 && "Repeat generator must repeat at least once");
        }

        T const& get() const override {
            if (m_current_repeat == 0) {
                m_returned.push_back(m_generator.get());
                return m_returned.back();
            }
            return m_returned[m_repeat_index];
        }

        bool next() override {
            // There are 2 basic cases:
            // 1) We are still reading the generator
            // 2) We are reading our own cache

            // In the first case, we need to poke the underlying generator.
            // If it happily moves, we are left in that state, otherwise it is time to start reading
from our cache
            if (m_current_repeat == 0) {
                const auto success = m_generator.next();
                if (!success) {
                    ++m_current_repeat;
                }
                return m_current_repeat < m_target_repeats;
            }

            // In the second case, we need to move indices forward and check that we haven't run up
against the end
            ++m_repeat_index;
            if (m_repeat_index == m_returned.size()) {
                m_repeat_index = 0;
                ++m_current_repeat;
            }
            return m_current_repeat < m_target_repeats;
        }
    };

    template <typename T>
    GeneratorWrapper<T> repeat(size_t repeats, GeneratorWrapper<T>&& generator) {
        return GeneratorWrapper<T>(pf::make_unique<RepeatGenerator<T>>(repeats,
std::move(generator)));
    }

    template <typename T, typename U, typename Func>
    class MapGenerator : public IGenerator<T> {
        // TBD: provide static assert for mapping function, for friendly error message
        GeneratorWrapper<U> m_generator;
        Func m_function;
        // To avoid returning dangling reference, we have to save the values
        T m_cache;
    public:
        template <typename F2 = Func>
        MapGenerator(F2&& function, GeneratorWrapper<U>&& generator) :
            m_generator(std::move(generator)),
            m_function(std::forward<F2>(function)),
            m_cache(m_function(m_generator.get()))
        {}

        T const& get() const override {
            return m_cache;
        }
        bool next() override {
            const auto success = m_generator.next();
            if (success) {
                m_cache = m_function(m_generator.get());
            }
            return success;
        }
    };

    template <typename Func, typename U, typename T = FunctionReturnType<Func, U»
    GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
        return GeneratorWrapper<T>(
            pf::make_unique<MapGenerator<T, U, Func>>(std::forward<Func>(function),
std::move(generator))
        );
    }

    template <typename T, typename U, typename Func>
    GeneratorWrapper<T> map(Func&& function, GeneratorWrapper<U>&& generator) {
        return GeneratorWrapper<T>(
            pf::make_unique<MapGenerator<T, U, Func>>(std::forward<Func>(function),
std::move(generator))
        );
    }

    template <typename T>
    class ChunkGenerator final : public IGenerator<std::vector<T» {
        std::vector<T> m_chunk;
        size_t m_chunk_size;
        GeneratorWrapper<T> m_generator;
        bool m_used_up = false;
    public:
        ChunkGenerator(size_t size, GeneratorWrapper<T> generator) :
            m_chunk_size(size), m_generator(std::move(generator))
        {
            m_chunk.reserve(m_chunk_size);
            if (m_chunk_size != 0) {
                m_chunk.push_back(m_generator.get());
                for (size_t i = 1; i < m_chunk_size; ++i) {
                    if (!m_generator.next()) {
                        Catch::throw_exception(GeneratorException("Not enough values to initialize the
first chunk"));
                    }
                    m_chunk.push_back(m_generator.get());
                }
            }
        }
        std::vector<T> const& get() const override {
            return m_chunk;
        }
        bool next() override {
            m_chunk.clear();
            for (size_t idx = 0; idx < m_chunk_size; ++idx) {
                if (!m_generator.next()) {
                    return false;
                }
                m_chunk.push_back(m_generator.get());
            }
            return true;
        }
    };

    template <typename T>
    GeneratorWrapper<std::vector<T>> chunk(size_t size, GeneratorWrapper<T>&& generator) {
        return GeneratorWrapper<std::vector<T>>(
            pf::make_unique<ChunkGenerator<T»(size, std::move(generator))
        );
    }

```

```
} // namespace Generators
} // namespace Catch

// end catch_generators_generic.hpp
// start catch_generators_specific.hpp

// start catch_context.h

#include <memory>

namespace Catch {

    struct IResultCapture;
    struct IRunner;
    struct IConfig;
    struct IMutableContext;

    using IConfigPtr = std::shared_ptr<IConfig const>;

    struct IContext
    {
        virtual ~IContext();

        virtual IResultCapture* getResultCapture() = 0;
        virtual IRunner* getRunner() = 0;
        virtual IConfigPtr const& getConfig() const = 0;
    };

    struct IMutableContext : IContext
    {
        virtual ~IMutableContext();
        virtual void setResultCapture( IResultCapture* resultCapture ) = 0;
        virtual void setRunner( IRunner* runner ) = 0;
        virtual void setConfig( IConfigPtr const& config ) = 0;

    private:
        static IMutableContext *currentContext;
        friend IMutableContext& getCurrentMutableContext();
        friend void cleanUpContext();
        static void createContext();
    };

    inline IMutableContext& getCurrentMutableContext()
    {
        if( !IMutableContext::currentContext )
            IMutableContext::createContext();
        // NOLINTNEXTLINE(clang-analyzer-core.uninitialized.UndefReturn)
        return *IMutableContext::currentContext;
    }

    inline IContext& getCurrentContext()
    {
        return getCurrentMutableContext();
    }

    void cleanUpContext();

    class SimplePcg32;
    SimplePcg32& rng();
}

// end catch_context.h
// start catch_interfaces_config.h

// start catch_option.hpp

namespace Catch {

    // An optional type
    template<typename T>
    class Option {
    public:
        Option() : nullableValue( nullptr ) {}
        Option( T const& _value )
        : nullableValue( new( storage ) T( _value ) )
        {}
        Option( Option const& _other )
        : nullableValue( _other ? new( storage ) T( *_other ) : nullptr )
        {}

        ~Option() {
            reset();
        }

        Option& operator= ( Option const& _other ) {
            if( &_other != this ) {
                reset();
                if( _other )
                    nullableValue = new( storage ) T( *_other );
            }
            return *this;
        }
        Option& operator = ( T const& _value ) {
            reset();
            nullableValue = new( storage ) T( _value );
            return *this;
        }

        void reset() {
            if( nullableValue )
                nullableValue->~T();
            nullableValue = nullptr;
        }

        T& operator*() { return *nullableValue; }
        T const& operator*() const { return *nullableValue; }
        T* operator->() { return nullableValue; }
        const T* operator->() const { return nullableValue; }

        T valueOr( T const& defaultValue ) const {
            return nullableValue ? *nullableValue : defaultValue;
        }

        bool some() const { return nullableValue != nullptr; }
        bool none() const { return nullableValue == nullptr; }

        bool operator !() const { return nullableValue == nullptr; }
        explicit operator bool() const {
            return some();
        }

    private:
        T *nullableValue;
        alignas(alignof(T)) char storage[sizeof(T)];
    };

} // end namespace Catch

// end catch_option.hpp
#include <chrono>
#include <iosfwd>
#include <string>
#include <vector>
#include <memory>

namespace Catch {

    enum class Verbosity {
        Quiet = 0,
        Normal,
        High
    };

    struct WarnAbout { enum What {
        Nothing = 0x00,
        NoAssertions = 0x01,
        NoTests = 0x02
    }; };

    struct ShowDurations { enum OrNot {
        DefaultForReporter,
        Always,
        Never
    }; };
    struct RunTests { enum InWhatOrder {
        InDeclarationOrder,
        InLexicographicalOrder,
        InRandomOrder
    }; };
    struct UseColour { enum YesOrNo {
        Auto,
        Yes,
        No
    }; };
    struct WaitForKeypress { enum When {
        Never,
        BeforeStart = 1,
        BeforeExit = 2,
        BeforeStartAndExit = BeforeStart | BeforeExit
    }; };

    class TestSpec;

    struct IConfig : NonCopyable {

        virtual ~IConfig();

        virtual bool allowThrows() const = 0;
        virtual std::ostream& stream() const = 0;
        virtual std::string name() const = 0;
        virtual bool includeSuccessfulResults() const = 0;
        virtual bool shouldDebugBreak() const = 0;
        virtual bool warnAboutMissingAssertions() const = 0;
        virtual bool warnAboutNoTests() const = 0;
        virtual int abortAfter() const = 0;
        virtual bool showInvisibles() const = 0;
        virtual ShowDurations::OrNot showDurations() const = 0;
        virtual double minDuration() const = 0;
        virtual TestSpec const& testSpec() const = 0;
        virtual bool hasTestFilters() const = 0;
        virtual std::vector<std::string> const& getTestsOrTags() const = 0;
        virtual RunTests::InWhatOrder runOrder() const = 0;
        virtual unsigned int rngSeed() const = 0;
        virtual UseColour::YesOrNo useColour() const = 0;
        virtual std::vector<std::string> const& getSectionsToRun() const = 0;
        virtual Verbosity verbosity() const = 0;

        virtual bool benchmarkNoAnalysis() const = 0;
        virtual int benchmarkSamples() const = 0;
        virtual double benchmarkConfidenceInterval() const = 0;
        virtual unsigned int benchmarkResamples() const = 0;
        virtual std::chrono::milliseconds benchmarkWarmupTime() const = 0;
    };

    using IConfigPtr = std::shared_ptr<IConfig const>;
}

// end catch_interfaces_config.h
// start catch_random_number_generator.h

#include <cstdint>

namespace Catch {

    // This is a simple implementation of C++11 Uniform Random Number
    // Generator. It does not provide all operators, because Catch2
    // does not use it, but it should behave as expected inside stdlib's
    // distributions.
    // The implementation is based on the PCG family (http://pcg-random.org)
    class SimplePcg32 {
        using state_type = std::uint64_t;
    public:
        using result_type = std::uint32_t;
        static constexpr result_type (min)() {
            return 0;
        }
        static constexpr result_type (max)() {
            return static_cast<result_type>(-1);
        }

        // Provide some default initial state for the default constructor
        SimplePcg32():SimplePcg32(0xed743cc4U) {}

        explicit SimplePcg32(result_type seed_);

        void seed(result_type seed_);
        void discard(uint64_t skip);

        result_type operator()();

    private:
        friend bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs);
        friend bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs);

        // In theory we also need operator« and operator»
        // In practice we do not use them, so we will skip them for now

        std::uint64_t m_state;
        // This part of the state determines which "stream" of the numbers
        // is chosen -- we take it as a constant for Catch2, so we only
        // need to deal with seeding the main state.
        // Picked by reading 8 bytes from `/dev/random` :-)
        static const std::uint64_t s_inc = (0x13ed0cc53f939476ULL « 1ULL) | 1ULL;
    };

} // end namespace Catch

// end catch_random_number_generator.h
#include <random>

namespace Catch {
```

```
namespace Generators {

template <typename Float>
class RandomFloatingGenerator final : public IGenerator<Float> {
    Catch::SimplePcg32& m_rng;
    std::uniform_real_distribution<Float> m_dist;
    Float m_current_number;
public:

    RandomFloatingGenerator(Float a, Float b):
        m_rng(rng()),
        m_dist(a, b) {
        static_cast<void>(next());
    }

    Float const& get() const override {
        return m_current_number;
    }
    bool next() override {
        m_current_number = m_dist(m_rng);
        return true;
    }
};

template <typename Integer>
class RandomIntegerGenerator final : public IGenerator<Integer> {
    Catch::SimplePcg32& m_rng;
    std::uniform_int_distribution<Integer> m_dist;
    Integer m_current_number;
public:

    RandomIntegerGenerator(Integer a, Integer b):
        m_rng(rng()),
        m_dist(a, b) {
        static_cast<void>(next());
    }

    Integer const& get() const override {
        return m_current_number;
    }
    bool next() override {
        m_current_number = m_dist(m_rng);
        return true;
    }
};

// TODO: Ideally this would be also constrained against the various char types,
//       but I don't expect users to run into that in practice.
template <typename T>
typename std::enable_if<std::is_integral<T>::value && !std::is_same<T, bool>::value,
GeneratorWrapper<T>>::type
random(T a, T b) {
    return GeneratorWrapper<T>(
        pf::make_unique<RandomIntegerGenerator<T>>(a, b)
    );
}

template <typename T>
typename std::enable_if<std::is_floating_point<T>::value,
GeneratorWrapper<T>>::type
random(T a, T b) {
    return GeneratorWrapper<T>(
        pf::make_unique<RandomFloatingGenerator<T>>(a, b)
    );
}

template <typename T>
class RangeGenerator final : public IGenerator<T> {
    T m_current;
    T m_end;
    T m_step;
    bool m_positive;

public:
    RangeGenerator(T const& start, T const& end, T const& step):
        m_current(start),
        m_end(end),
        m_step(step),
        m_positive(m_step > T(0))
    {
        assert(m_current != m_end && "Range start and end cannot be equal");
        assert(m_step != T(0) && "Step size cannot be zero");
        assert(((m_positive && m_current <= m_end) || (!m_positive && m_current >= m_end)) && "Step
moves away from end");
    }

    RangeGenerator(T const& start, T const& end):
        RangeGenerator(start, end, (start < end) ? T(1) : T(-1))
    {}

    T const& get() const override {
        return m_current;
    }

    bool next() override {
        m_current += m_step;
        return (m_positive) ? (m_current < m_end) : (m_current > m_end);
    }
};

template <typename T>
GeneratorWrapper<T> range(T const& start, T const& end, T const& step) {
    static_assert(std::is_arithmetic<T>::value && !std::is_same<T, bool>::value, "Type must be
numeric");
    return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end, step));
}

template <typename T>
GeneratorWrapper<T> range(T const& start, T const& end) {
    static_assert(std::is_integral<T>::value && !std::is_same<T, bool>::value, "Type must be an
integer");
    return GeneratorWrapper<T>(pf::make_unique<RangeGenerator<T>>(start, end));
}

template <typename T>
class IteratorGenerator final : public IGenerator<T> {
    static_assert(!std::is_same<T, bool>::value,
        "IteratorGenerator currently does not support bools"
        "because of std::vector<bool> specialization");

    std::vector<T> m_elems;
    size_t m_current = 0;
public:
    template <typename InputIterator, typename InputSentinel>
    IteratorGenerator(InputIterator first, InputSentinel last):m_elems(first, last) {
        if (m_elems.empty()) {
            Catch::throw_exception(GeneratorException("IteratorGenerator received no valid values"));
        }
    }

    T const& get() const override {
        return m_elems[m_current];
    }

    bool next() override {
        ++m_current;
        return m_current != m_elems.size();
    }
};

template <typename InputIterator,
          typename InputSentinel,
          typename ResultType = typename std::iterator_traits<InputIterator>::value_type>
GeneratorWrapper<ResultType> from_range(InputIterator from, InputSentinel to) {
    return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(from, to));
}

template <typename Container,
          typename ResultType = typename Container::value_type>
GeneratorWrapper<ResultType> from_range(Container const& cnt) {
    return GeneratorWrapper<ResultType>(pf::make_unique<IteratorGenerator<ResultType>>(cnt.begin(),
cnt.end()));
}

} // namespace Generators
} // namespace Catch

// end catch_generators_specific.hpp

// These files are included here so the single_include script doesn't put them
// in the conditionally compiled sections
// start catch_test_case_info.h

#include <string>
#include <vector>
#include <memory>

#ifdef __clang__
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wpadded"
#endif

namespace Catch {

    struct ITestInvoker;

    struct TestCaseInfo {
        enum SpecialProperties{
            None = 0,
            IsHidden = 1 « 1,
            ShouldFail = 1 « 2,
            MayFail = 1 « 3,
            Throws = 1 « 4,
            NonPortable = 1 « 5,
            Benchmark = 1 « 6
        };

        TestCaseInfo(   std::string const& _name,
                        std::string const& _className,
                        std::string const& _description,
                        std::vector<std::string> const& _tags,
                        SourceLineInfo const& _lineInfo );

        friend void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags );

        bool isHidden() const;
        bool throws() const;
        bool okToFail() const;
        bool expectedToFail() const;

        std::string tagsAsString() const;

        std::string name;
        std::string className;
        std::string description;
        std::vector<std::string> tags;
        std::vector<std::string> lcaseTags;
        SourceLineInfo lineInfo;
        SpecialProperties properties;
    };

    class TestCase : public TestCaseInfo {
    public:

        TestCase( ITestInvoker* testCase, TestCaseInfo&& info );

        TestCase withName( std::string const& _newName ) const;

        void invoke() const;

        TestCaseInfo const& getTestCaseInfo() const;

        bool operator == ( TestCase const& other ) const;
        bool operator < ( TestCase const& other ) const;

    private:
        std::shared_ptr<ITestInvoker> test;
    };

    TestCase makeTestCase(  ITestInvoker* testCase,
                            std::string const& className,
                            NameAndTags const& nameAndTags,
                            SourceLineInfo const& lineInfo );
}

#ifdef __clang__
#pragma clang diagnostic pop
#endif

// end catch_test_case_info.h
// start catch_interfaces_runner.h

namespace Catch {

    struct IRunner {
        virtual ~IRunner();
        virtual bool aborting() const = 0;
    };
}

// end catch_interfaces_runner.h

#ifdef __OBJC__
// start catch_objc.hpp

#import <objc/runtime.h>

#include <string>

// NB. Any general catch headers included here must be included
// in catch.hpp first to make sure they are included by the single
// header for non obj-usage

// This protocol is really only here for (self) documenting purposes, since
// all its methods are optional.
@protocol OcFixture

@optional

-(void) setUp;
-(void) tearDown;

@end

namespace Catch {

    class OcMethod : public ITestInvoker {

    public:
        OcMethod( Class cls, SEL sel ) : m_cls( cls ), m_sel( sel ) {}

        virtual void invoke() const {
            id obj = [[m_cls alloc] init];

            performOptionalSelector( obj, @selector(setUp)  );
            performOptionalSelector( obj, m_sel );
            performOptionalSelector( obj, @selector(tearDown)  );

            arcSafeRelease( obj );
        }
    private:
        virtual ~OcMethod() {}

        Class m_cls;
        SEL m_sel;
    };

    namespace Detail{

        inline std::string getAnnotation(   Class cls,
                                            std::string const& annotationName,
                                            std::string const& testCaseName ) {
            NSString* selStr = [[NSString alloc] initWithFormat:@"Catch_%s_%s",
annotationName.c_str(), testCaseName.c_str()];
            SEL sel = NSSelectorFromString( selStr );
            arcSafeRelease( selStr );
            id value = performOptionalSelector( cls, sel );
            if( value )
                return [(NSString*)value UTF8String];
            return "";
        }
    }

    inline std::size_t registerTestMethods() {
        std::size_t noTestMethods = 0;
        int noClasses = objc_getClassList( nullptr, 0 );

        Class* classes = (CATCH_UNSAFE_UNRETAINED Class *)malloc( sizeof(Class) * noClasses);
        objc_getClassList( classes, noClasses );

        for( int c = 0; c < noClasses; c++ ) {
            Class cls = classes[c];
            {
                u_int count;
                Method* methods = class_copyMethodList( cls, &count );
                for( u_int m = 0; m < count ; m++ ) {
                    SEL selector = method_getName(methods[m]);
                    std::string methodName = sel_getName(selector);
                    if( startsWith( methodName, "Catch_TestCase_" ) ) {
                        std::string testCaseName = methodName.substr( 15 );
                        std::string name = Detail::getAnnotation( cls, "Name", testCaseName );
                        std::string desc = Detail::getAnnotation( cls, "Description", testCaseName );
                        const char* className = class_getName( cls );

                        getMutableRegistryHub().registerTest( makeTestCase( new OcMethod( cls,
selector ), className, NameAndTags( name.c_str(), desc.c_str() ), SourceLineInfo("",0) ) );
                        noTestMethods++;
                    }
                }
                free(methods);
            }
        }
        return noTestMethods;
    }

#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)

    namespace Matchers {
```

```
    namespace Impl {
    namespace NSStringMatchers {

        struct StringHolder : MatcherBase<NSString*>{
            StringHolder( NSString* substr ) : m_substr( [substr copy] ){}
            StringHolder( StringHolder const& other ) : m_substr( [other.m_substr copy] ){}
            StringHolder() {
                arcSafeRelease( m_substr );
            }

            bool match( NSString* str ) const override {
                return false;
            }

            NSString* CATCH_ARC_STRONG m_substr;
        };

        struct Equals : StringHolder {
            Equals( NSString* substr ) : StringHolder( substr ){}

            bool match( NSString* str ) const override {
                return  (str != nil || m_substr == nil ) &&
                        [str isEqualToString:m_substr];
            }

            std::string describe() const override {
                return "equals string: " + Catch::Detail::stringify( m_substr );
            }
        };

        struct Contains : StringHolder {
            Contains( NSString* substr ) : StringHolder( substr ){}

            bool match( NSString* str ) const override {
                return  (str != nil || m_substr == nil ) &&
                        [str rangeOfString:m_substr].location != NSNotFound;
            }

            std::string describe() const override {
                return "contains string: " + Catch::Detail::stringify( m_substr );
            }
        };

        struct StartsWith : StringHolder {
            StartsWith( NSString* substr ) : StringHolder( substr ){}

            bool match( NSString* str ) const override {
                return  (str != nil || m_substr == nil ) &&
                        [str rangeOfString:m_substr].location == 0;
            }

            std::string describe() const override {
                return "starts with: " + Catch::Detail::stringify( m_substr );
            }
        };
        struct EndsWith : StringHolder {
            EndsWith( NSString* substr ) : StringHolder( substr ){}

            bool match( NSString* str ) const override {
                return  (str != nil || m_substr == nil ) &&
                        [str rangeOfString:m_substr].location == [str length] - [m_substr length];
            }

            std::string describe() const override {
                return "ends with: " + Catch::Detail::stringify( m_substr );
            }
        };

    } // namespace NSStringMatchers
    } // namespace Impl

    inline Impl::NSStringMatchers::Equals
        Equals( NSString* substr ){ return Impl::NSStringMatchers::Equals( substr ); }

    inline Impl::NSStringMatchers::Contains
        Contains( NSString* substr ){ return Impl::NSStringMatchers::Contains( substr ); }

    inline Impl::NSStringMatchers::StartsWith
        StartsWith( NSString* substr ){ return Impl::NSStringMatchers::StartsWith( substr ); }

    inline Impl::NSStringMatchers::EndsWith
        EndsWith( NSString* substr ){ return Impl::NSStringMatchers::EndsWith( substr ); }

} // namespace Matchers

using namespace Matchers;

```

```
#endif // CATCH_CONFIG_DISABLE_MATCHERS

} // namespace Catch

#define OC_MAKE_UNIQUE_NAME( root, uniqueSuffix ) root##uniqueSuffix
#define OC_TEST_CASE2( name, desc, uniqueSuffix ) \
+(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Name_test_, uniqueSuffix ) \
{ \
return @ name; \
} \
+(NSString*) OC_MAKE_UNIQUE_NAME( Catch_Description_test_, uniqueSuffix ) \
{ \
return @ desc; \
} \
-(void) OC_MAKE_UNIQUE_NAME( Catch_TestCase_test_, uniqueSuffix )

#define OC_TEST_CASE( name, desc ) OC_TEST_CASE2( name, desc, __LINE__ )

// end catch_objc.hpp
#endif

// Benchmarking needs the externally-facing parts of reporters to work
#if defined(CATCH_CONFIG_EXTERNAL_INTERFACES) || defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
// start catch_external_interfaces.h

// start catch_reporter_bases.hpp

// start catch_interfaces_reporter.h

// start catch_config.hpp

// start catch_test_spec_parser.h

#ifdef __clang__
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wpadded"
#endif

// start catch_test_spec.h

#ifdef __clang__
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wpadded"
#endif

// start catch_wildcard_pattern.h

namespace Catch
{
    class WildcardPattern {
        enum WildcardPosition {
            NoWildcard = 0,
            WildcardAtStart = 1,
            WildcardAtEnd = 2,
            WildcardAtBothEnds = WildcardAtStart | WildcardAtEnd
        };

    public:

        WildcardPattern( std::string const& pattern, CaseSensitive::Choice caseSensitivity );
        virtual ~WildcardPattern() = default;
        virtual bool matches( std::string const& str ) const;

    private:
        std::string normaliseString( std::string const& str ) const;
        CaseSensitive::Choice m_caseSensitivity;
        WildcardPosition m_wildcard = NoWildcard;
        std::string m_pattern;
    };
}

// end catch_wildcard_pattern.h
#include <string>
#include <vector>
#include <memory>

namespace Catch {

    struct IConfig;

    class TestSpec {
        class Pattern {
        public:
            explicit Pattern( std::string const& name );
            virtual ~Pattern();
            virtual bool matches( TestCaseInfo const& testCase ) const = 0;
            std::string const& name() const;
        private:
            std::string const m_name;
        };
        using PatternPtr = std::shared_ptr<Pattern>;

        class NamePattern : public Pattern {
        public:
            explicit NamePattern( std::string const& name, std::string const& filterString );
            bool matches( TestCaseInfo const& testCase ) const override;
        private:
            WildcardPattern m_wildcardPattern;
        };

        class TagPattern : public Pattern {
        public:
            explicit TagPattern( std::string const& tag, std::string const& filterString );
            bool matches( TestCaseInfo const& testCase ) const override;
        private:
            std::string m_tag;
        };

        class ExcludedPattern : public Pattern {
        public:
            explicit ExcludedPattern( PatternPtr const& underlyingPattern );
            bool matches( TestCaseInfo const& testCase ) const override;
        private:
            PatternPtr m_underlyingPattern;
        };

        struct Filter {
            std::vector<PatternPtr> m_patterns;

            bool matches( TestCaseInfo const& testCase ) const;
            std::string name() const;
        };

    public:
        struct FilterMatch {
            std::string name;
            std::vector<TestCase const*> tests;
        };
        using Matches = std::vector<FilterMatch>;
        using vectorStrings = std::vector<std::string>;

        bool hasFilters() const;
        bool matches( TestCaseInfo const& testCase ) const;
        Matches matchesByFilter( std::vector<TestCase> const& testCases, IConfig const& config )
const;
        const vectorStrings & getInvalidArgs() const;

    private:
        std::vector<Filter> m_filters;
        std::vector<std::string> m_invalidArgs;
        friend class TestSpecParser;
    };
}

#ifdef __clang__
#pragma clang diagnostic pop
#endif

// end catch_test_spec.h
// start catch_interfaces_tag_alias_registry.h

#include <string>

namespace Catch {

    struct TagAlias;

    struct ITagAliasRegistry {
        virtual ~ITagAliasRegistry();
        // Nullptr if not present
        virtual TagAlias const* find( std::string const& alias ) const = 0;
        virtual std::string expandAliases( std::string const& unexpandedTestSpec ) const = 0;

        static ITagAliasRegistry const& get();
    };

} // end namespace Catch

// end catch_interfaces_tag_alias_registry.h
namespace Catch {

    class TestSpecParser {
        enum Mode{ None, Name, QuotedName, Tag, EscapedName };
        Mode m_mode = None;
        Mode lastMode = None;
        bool m_exclusion = false;
        std::size_t m_pos = 0;
        std::size_t m_realPatternPos = 0;
        std::string m_arg;
        std::string m_substring;
        std::string m_patternName;
        std::vector<std::size_t> m_escapeChars;
        TestSpec::Filter m_currentFilter;
        TestSpec m_testSpec;
        ITagAliasRegistry const* m_tagAliases = nullptr;

    public:
        TestSpecParser( ITagAliasRegistry const& tagAliases );

        TestSpecParser& parse( std::string const& arg );
        TestSpec testSpec();

    private:
        bool visitChar( char c );
        void startNewMode( Mode mode );
        bool processNoneChar( char c );
        void processNameChar( char c );
        bool processOtherChar( char c );
        void endMode();
        void escape();
        bool isControlChar( char c ) const;
        void saveLastMode();
        void revertBackToLastMode();
        void addFilter();
        bool separate();

        // Handles common preprocessing of the pattern for name/tag patterns
        std::string preprocessPattern();
        // Adds the current pattern as a test name
        void addNamePattern();
        // Adds the current pattern as a tag
        void addTagPattern();

        inline void addCharToPattern(char c) {
            m_substring += c;
            m_patternName += c;
            m_realPatternPos++;
        }

    };
    TestSpec parseTestSpec( std::string const& arg );

} // namespace Catch

#ifdef __clang__
#pragma clang diagnostic pop
#endif

// end catch_test_spec_parser.h
// Libstdc++ doesn't like incomplete classes for unique_ptr

#include <memory>
#include <vector>
#include <string>

#ifndef CATCH_CONFIG_CONSOLE_WIDTH
#define CATCH_CONFIG_CONSOLE_WIDTH 80
#endif

namespace Catch {

    struct IStream;

    struct ConfigData {
        bool listTests = false;
        bool listTags = false;
        bool listReporters = false;
        bool listTestNamesOnly = false;

        bool showSuccessfulTests = false;
        bool shouldDebugBreak = false;
        bool noThrow = false;
        bool showHelp = false;
        bool showInvisibles = false;
        bool filenamesAsTags = false;
        bool libIdentify = false;

        int abortAfter = -1;
        unsigned int rngSeed = 0;

        bool benchmarkNoAnalysis = false;
        unsigned int benchmarkSamples = 100;
        double benchmarkConfidenceInterval = 0.95;
        unsigned int benchmarkResamples = 100000;
        std::chrono::milliseconds::rep benchmarkWarmupTime = 100;

        Verbosity verbosity = Verbosity::Normal;
        WarnAbout::What warnings = WarnAbout::Nothing;
        ShowDurations::OrNot showDurations = ShowDurations::DefaultForReporter;
        double minDuration = -1;
        RunTests::InWhatOrder runOrder = RunTests::InDeclarationOrder;
        UseColour::YesOrNo useColour = UseColour::Auto;
        WaitForKeypress::When waitForKeypress = WaitForKeypress::Never;

        std::string outputFilename;
        std::string name;
        std::string processName;
#ifndef CATCH_CONFIG_DEFAULT_REPORTER
#define CATCH_CONFIG_DEFAULT_REPORTER "console"
#endif
        std::string reporterName = CATCH_CONFIG_DEFAULT_REPORTER;
#undef CATCH_CONFIG_DEFAULT_REPORTER

        std::vector<std::string> testsOrTags;
        std::vector<std::string> sectionsToRun;
    };

    class Config : public IConfig {
    public:

        Config() = default;
        Config( ConfigData const& data );
        virtual ~Config() = default;

        std::string const& getFilename() const;

        bool listTests() const;
        bool listTestNamesOnly() const;
        bool listTags() const;
        bool listReporters() const;

        std::string getProcessName() const;
        std::string const& getReporterName() const;

        std::vector<std::string> const& getTestsOrTags() const override;
        std::vector<std::string> const& getSectionsToRun() const override;

        TestSpec const& testSpec() const override;
        bool hasTestFilters() const override;

        bool showHelp() const;

        // IConfig interface
        bool allowThrows() const override;
        std::ostream& stream() const override;
        std::string name() const override;
        bool includeSuccessfulResults() const override;
        bool warnAboutMissingAssertions() const override;
        bool warnAboutNoTests() const override;
        ShowDurations::OrNot showDurations() const override;
        double minDuration() const override;
        RunTests::InWhatOrder runOrder() const override;
        unsigned int rngSeed() const override;
        UseColour::YesOrNo useColour() const override;
        bool shouldDebugBreak() const override;
        int abortAfter() const override;
        bool showInvisibles() const override;
        Verbosity verbosity() const override;
        bool benchmarkNoAnalysis() const override;
        int benchmarkSamples() const override;
        double benchmarkConfidenceInterval() const override;
        unsigned int benchmarkResamples() const override;
        std::chrono::milliseconds benchmarkWarmupTime() const override;

    private:

        IStream const* openStream();
        ConfigData m_data;

        std::unique_ptr<IStream const> m_stream;
        TestSpec m_testSpec;
        bool m_hasTestFilters = false;
    };

} // end namespace Catch

// end catch_config.hpp
// start catch_assertionresult.h
```

```

#include <string>

namespace Catch {

    struct AssertionResultData
    {
        AssertionResultData() = delete;

        AssertionResultData( ResultWas::OfType _resultType, LazyExpression const& _lazyExpression );

        std::string message;
        mutable std::string reconstructedExpression;
        LazyExpression lazyExpression;
        ResultWas::OfType resultType;

        std::string reconstructExpression() const;
    };

    class AssertionResult {
    public:
        AssertionResult() = delete;
        AssertionResult( AssertionInfo const& info, AssertionResultData const& data );

        bool isOk() const;
        bool succeeded() const;
        ResultWas::OfType getResultType() const;
        bool hasExpression() const;
        bool hasMessage() const;
        std::string getExpression() const;
        std::string getExpressionInMacro() const;
        bool hasExpandedExpression() const;
        std::string getExpandedExpression() const;
        std::string getMessage() const;
        SourceLineInfo getSourceInfo() const;
        StringRef getTestMacroName() const;

    //protected:
        AssertionInfo m_info;
        AssertionResultData m_resultData;
    };

} // end namespace Catch

// end catch_assertionresult.h
#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
// start catch_estimate.hpp

 // Statistics estimates


namespace Catch {
    namespace Benchmark {
        template <typename Duration>
        struct Estimate {
            Duration point;
            Duration lower_bound;
            Duration upper_bound;
            double confidence_interval;

            template <typename Duration2>
            operator Estimate<Duration2>() const {
                return { point, lower_bound, upper_bound, confidence_interval };
            }
        };
    } // namespace Benchmark
} // namespace Catch

// end catch_estimate.hpp
// start catch_outlier_classification.hpp

// Outlier information

namespace Catch {
    namespace Benchmark {
        struct OutlierClassification {
            int samples_seen = 0;
            int low_severe = 0;     // more than 3 times IQR below Q1
            int low_mild = 0;       // 1.5 to 3 times IQR below Q1
            int high_mild = 0;      // 1.5 to 3 times IQR above Q3
            int high_severe = 0;    // more than 3 times IQR above Q3

            int total() const {
                return low_severe + low_mild + high_mild + high_severe;
            }
        };
    } // namespace Benchmark
} // namespace Catch

// end catch_outlier_classification.hpp

#include <iterator>
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

#include <string>
#include <iosfwd>
#include <map>
#include <set>
#include <memory>
#include <algorithm>

namespace Catch {

    struct ReporterConfig {
        explicit ReporterConfig( IConfigPtr const& _fullConfig );

        ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream );

        std::ostream& stream() const;
        IConfigPtr fullConfig() const;

    private:
        std::ostream* m_stream;
        IConfigPtr m_fullConfig;
    };

    struct ReporterPreferences {
        bool shouldRedirectStdOut = false;
        bool shouldReportAllAssertions = false;
    };

    template<typename T>
    struct LazyStat : Option<T> {
        LazyStat& operator=( T const& _value ) {
            Option<T>::operator=( _value );
            used = false;
            return *this;
        }
        void reset() {
            Option<T>::reset();
            used = false;
        }
        bool used = false;
    };

    struct TestRunInfo {
        TestRunInfo( std::string const& _name );
        std::string name;
    };
    struct GroupInfo {
        GroupInfo(  std::string const& _name,
                    std::size_t _groupIndex,
                    std::size_t _groupsCount );

        std::string name;
        std::size_t groupIndex;
        std::size_t groupsCounts;
    };

    struct AssertionStats {
        AssertionStats( AssertionResult const& _assertionResult,
                        std::vector<MessageInfo> const& _infoMessages,
                        Totals const& _totals );

        AssertionStats( AssertionStats const& )              = default;
        AssertionStats( AssertionStats && )                  = default;
        AssertionStats& operator = ( AssertionStats const& ) = delete;
        AssertionStats& operator = ( AssertionStats && )     = delete;
        virtual ~AssertionStats();

        AssertionResult assertionResult;
        std::vector<MessageInfo> infoMessages;
        Totals totals;
    };

    struct SectionStats {
        SectionStats(   SectionInfo const& _sectionInfo,
                        Counts const& _assertions,
                        double _durationInSeconds,
                        bool _missingAssertions );
        SectionStats( SectionStats const& )              = default;
        SectionStats( SectionStats && )                  = default;
        SectionStats& operator = ( SectionStats const& ) = default;
        SectionStats& operator = ( SectionStats && )     = default;
        virtual ~SectionStats();

        SectionInfo sectionInfo;
        Counts assertions;
        double durationInSeconds;
        bool missingAssertions;
    };

    struct TestCaseStats {
        TestCaseStats(  TestCaseInfo const& _testInfo,
                        Totals const& _totals,
                        std::string const& _stdOut,
                        std::string const& _stdErr,
                        bool _aborting );

        TestCaseStats( TestCaseStats const& )              = default;
        TestCaseStats( TestCaseStats && )                  = default;
        TestCaseStats& operator = ( TestCaseStats const& ) = default;
        TestCaseStats& operator = ( TestCaseStats && )     = default;
        virtual ~TestCaseStats();

        TestCaseInfo testInfo;
        Totals totals;
        std::string stdOut;
        std::string stdErr;
        bool aborting;
    };

    struct TestGroupStats {
        TestGroupStats( GroupInfo const& _groupInfo,
                        Totals const& _totals,
                        bool _aborting );
        TestGroupStats( GroupInfo const& _groupInfo );

        TestGroupStats( TestGroupStats const& )              = default;
        TestGroupStats( TestGroupStats && )                  = default;
        TestGroupStats& operator = ( TestGroupStats const& ) = default;
        TestGroupStats& operator = ( TestGroupStats && )     = default;
        virtual ~TestGroupStats();

        GroupInfo groupInfo;
        Totals totals;
        bool aborting;
    };

    struct TestRunStats {
        TestRunStats(   TestRunInfo const& _runInfo,
                        Totals const& _totals,
                        bool _aborting );

        TestRunStats( TestRunStats const& )              = default;
        TestRunStats( TestRunStats && )                  = default;
        TestRunStats& operator = ( TestRunStats const& ) = default;
        TestRunStats& operator = ( TestRunStats && )     = default;
        virtual ~TestRunStats();

        TestRunInfo runInfo;
        Totals totals;
        bool aborting;
    };

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
    struct BenchmarkInfo {
        std::string name;
        double estimatedDuration;
        int iterations;
        int samples;
        unsigned int resamples;
        double clockResolution;
        double clockCost;
    };

    template <class Duration>
    struct BenchmarkStats {
        BenchmarkInfo info;

        std::vector<Duration> samples;
        Benchmark::Estimate<Duration> mean;
        Benchmark::Estimate<Duration> standardDeviation;
        Benchmark::OutlierClassification outliers;
        double outlierVariance;

        template <typename Duration2>
        operator BenchmarkStats<Duration2>() const {
            std::vector<Duration2> samples2;
            samples2.reserve(samples.size());
            std::transform(samples.begin(), samples.end(), std::back_inserter(samples2), [](Duration
d) { return Duration2(d); });
            return {
                info,
                std::move(samples2),
                mean,
                standardDeviation,
                outliers,
                outlierVariance,
            };
        }
    };
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

    struct IStreamingReporter {
        virtual ~IStreamingReporter() = default;

        // Implementing class must also provide the following static methods:
        // static std::string getDescription();
        // static std::set<Verbosity> getSupportedVerbosities()

        virtual ReporterPreferences getPreferences() const = 0;

        virtual void noMatchingTestCases( std::string const& spec ) = 0;

        virtual void reportInvalidArguments(std::string const&) {}

        virtual void testRunStarting( TestRunInfo const& testRunInfo ) = 0;
        virtual void testGroupStarting( GroupInfo const& groupInfo ) = 0;

        virtual void testCaseStarting( TestCaseInfo const& testInfo ) = 0;
        virtual void sectionStarting( SectionInfo const& sectionInfo ) = 0;

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
        virtual void benchmarkPreparing( std::string const& ) {}
        virtual void benchmarkStarting( BenchmarkInfo const& ) {}
        virtual void benchmarkEnded( BenchmarkStats<> const& ) {}
        virtual void benchmarkFailed( std::string const& ) {}
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

        virtual void assertionStarting( AssertionInfo const& assertionInfo ) = 0;

        // The return value indicates if the messages buffer should be cleared:
        virtual bool assertionEnded( AssertionStats const& assertionStats ) = 0;

        virtual void sectionEnded( SectionStats const& sectionStats ) = 0;
        virtual void testCaseEnded( TestCaseStats const& testCaseStats ) = 0;
        virtual void testGroupEnded( TestGroupStats const& testGroupStats ) = 0;
        virtual void testRunEnded( TestRunStats const& testRunStats ) = 0;

        virtual void skipTest( TestCaseInfo const& testInfo ) = 0;

        // Default empty implementation provided
        virtual void fatalErrorEncountered( StringRef name );

        virtual bool isMulti() const;
    };
    using IStreamingReporterPtr = std::unique_ptr<IStreamingReporter>;

    struct IReporterFactory {
        virtual ~IReporterFactory();
        virtual IStreamingReporterPtr create( ReporterConfig const& config ) const = 0;
        virtual std::string getDescription() const = 0;
    };
    using IReporterFactoryPtr = std::shared_ptr<IReporterFactory>;

    struct IReporterRegistry {
        using FactoryMap = std::map<std::string, IReporterFactoryPtr>;
        using Listeners = std::vector<IReporterFactoryPtr>;

        virtual ~IReporterRegistry();
        virtual IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config )
const = 0;
        virtual FactoryMap const& getFactories() const = 0;
        virtual Listeners const& getListeners() const = 0;
    };

} // end namespace Catch

// end catch_interfaces_reporter.h
#include <algorithm>
#include <cstring>
#include <cfloat>
#include <cstdio>
#include <cassert>
#include <memory>
#include <ostream>

```

```
namespace Catch {
    void prepareExpandedExpression(AssertionResult& result);

    // Returns double formatted as %.3f (format expected on output)
    std::string getFormattedDuration( double duration );

    bool shouldShowDuration( IConfig const& config, double duration );

    std::string serializeFilters( std::vector<std::string> const& container );

    template<typename DerivedT>
    struct StreamingReporterBase : IStreamingReporter {

        StreamingReporterBase( ReporterConfig const& _config )
        :   m_config( _config.fullConfig() ),
            stream( _config.stream() )
        {
            m_reporterPrefs.shouldRedirectStdOut = false;
            if( !DerivedT::getSupportedVerbosities().count( m_config->verbosity() ) )
                CATCH_ERROR( "Verbosity level not supported by this reporter" );
        }

        ReporterPreferences getPreferences() const override {
            return m_reporterPrefs;
        }

        static std::set<Verbosity> getSupportedVerbosities() {
            return { Verbosity::Normal };
        }

        ~StreamingReporterBase() override = default;

        void noMatchingTestCases(std::string const&) override {}

        void reportInvalidArguments(std::string const&) override {}

        void testRunStarting(TestRunInfo const& _testRunInfo) override {
            currentTestRunInfo = _testRunInfo;
        }

        void testGroupStarting(GroupInfo const& _groupInfo) override {
            currentGroupInfo = _groupInfo;
        }

        void testCaseStarting(TestCaseInfo const& _testInfo) override  {
            currentTestCaseInfo = _testInfo;
        }
        void sectionStarting(SectionInfo const& _sectionInfo) override {
            m_sectionStack.push_back(_sectionInfo);
        }

        void sectionEnded(SectionStats const& /* _sectionStats */) override {
            m_sectionStack.pop_back();
        }
        void testCaseEnded(TestCaseStats const& /* _testCaseStats */) override {
            currentTestCaseInfo.reset();
        }
        void testGroupEnded(TestGroupStats const& /* _testGroupStats */) override {
            currentGroupInfo.reset();
        }
        void testRunEnded(TestRunStats const& /* _testRunStats */) override {
            currentTestCaseInfo.reset();
            currentGroupInfo.reset();
            currentTestRunInfo.reset();
        }

        void skipTest(TestCaseInfo const&) override {
            // Don't do anything with this by default.
            // It can optionally be overridden in the derived class.
        }

        IConfigPtr m_config;
        std::ostream& stream;

        LazyStat<TestRunInfo> currentTestRunInfo;
        LazyStat<GroupInfo> currentGroupInfo;
        LazyStat<TestCaseInfo> currentTestCaseInfo;

        std::vector<SectionInfo> m_sectionStack;
        ReporterPreferences m_reporterPrefs;
    };

    template<typename DerivedT>
    struct CumulativeReporterBase : IStreamingReporter {
        template<typename T, typename ChildNodeT>
        struct Node {
            explicit Node( T const& _value ) : value( _value ) {}
            virtual ~Node() {}

            using ChildNodes = std::vector<std::shared_ptr<ChildNodeT»;
            T value;
            ChildNodes children;
        };
        struct SectionNode {
            explicit SectionNode(SectionStats const& _stats) : stats(_stats) {}
            virtual ~SectionNode() = default;

            bool operator == (SectionNode const& other) const {
                return stats.sectionInfo.lineInfo == other.stats.sectionInfo.lineInfo;
            }
            bool operator == (std::shared_ptr<SectionNode> const& other) const {
                return operator==(*other);
            }

            SectionStats stats;
            using ChildSections = std::vector<std::shared_ptr<SectionNode»;
            using Assertions = std::vector<AssertionStats>;
            ChildSections childSections;
            Assertions assertions;
            std::string stdOut;
            std::string stdErr;
        };

        struct BySectionInfo {
            BySectionInfo( SectionInfo const& other ) : m_other( other ) {}
            BySectionInfo( BySectionInfo const& other ) : m_other( other.m_other ) {}
            bool operator() (std::shared_ptr<SectionNode> const& node) const {
                return ((node->stats.sectionInfo.name == m_other.name) &&
                        (node->stats.sectionInfo.lineInfo == m_other.lineInfo));
            }
            void operator=(BySectionInfo const&) = delete;

        private:
            SectionInfo const& m_other;
        };

        using TestCaseNode = Node<TestCaseStats, SectionNode>;
        using TestGroupNode = Node<TestGroupStats, TestCaseNode>;
        using TestRunNode = Node<TestRunStats, TestGroupNode>;

        CumulativeReporterBase( ReporterConfig const& _config )
        :   m_config( _config.fullConfig() ),
            stream( _config.stream() )
        {
            m_reporterPrefs.shouldRedirectStdOut = false;
            if( !DerivedT::getSupportedVerbosities().count( m_config->verbosity() ) )
                CATCH_ERROR( "Verbosity level not supported by this reporter" );
        }
        ~CumulativeReporterBase() override = default;

        ReporterPreferences getPreferences() const override {
            return m_reporterPrefs;
        }

        static std::set<Verbosity> getSupportedVerbosities() {
            return { Verbosity::Normal };
        }

        void testRunStarting( TestRunInfo const& ) override {}
        void testGroupStarting( GroupInfo const& ) override {}

        void testCaseStarting( TestCaseInfo const& ) override {}

        void sectionStarting( SectionInfo const& sectionInfo ) override {
            SectionStats incompleteStats( sectionInfo, Counts(), 0, false );
            std::shared_ptr<SectionNode> node;
            if( m_sectionStack.empty() ) {
                if( !m_rootSection )
                    m_rootSection = std::make_shared<SectionNode>( incompleteStats );
                node = m_rootSection;
            }
            else {
                SectionNode& parentNode = *m_sectionStack.back();
                auto it =
                    std::find_if(   parentNode.childSections.begin(),
                                    parentNode.childSections.end(),
                                    BySectionInfo( sectionInfo ) );
                if( it == parentNode.childSections.end() ) {
                    node = std::make_shared<SectionNode>( incompleteStats );
                    parentNode.childSections.push_back( node );
                }
                else
                    node = *it;
            }
            m_sectionStack.push_back( node );
            m_deepestSection = std::move(node);
        }

        void assertionStarting(AssertionInfo const&) override {}

        bool assertionEnded(AssertionStats const& assertionStats) override {
            assert(!m_sectionStack.empty());
            // AssertionResult holds a pointer to a temporary DecomposedExpression,
            // which getExpandedExpression() calls to build the expression string.
            // Our section stack copy of the assertionResult will likely outlive the
            // temporary, so it must be expanded or discarded now to avoid calling
            // a destroyed object later.
            prepareExpandedExpression(const_cast<AssertionResult&>( assertionStats.assertionResult )
);
            SectionNode& sectionNode = *m_sectionStack.back();
            sectionNode.assertions.push_back(assertionStats);
            return true;
        }
        void sectionEnded(SectionStats const& sectionStats) override {
            assert(!m_sectionStack.empty());
            SectionNode& node = *m_sectionStack.back();
            node.stats = sectionStats;
            m_sectionStack.pop_back();
        }
        void testCaseEnded(TestCaseStats const& testCaseStats) override {
            auto node = std::make_shared<TestCaseNode>(testCaseStats);
            assert(m_sectionStack.size() == 0);
            node->children.push_back(m_rootSection);
            m_testCases.push_back(node);
            m_rootSection.reset();

            assert(m_deepestSection);
            m_deepestSection->stdOut = testCaseStats.stdOut;
            m_deepestSection->stdErr = testCaseStats.stdErr;
        }
        void testGroupEnded(TestGroupStats const& testGroupStats) override {
            auto node = std::make_shared<TestGroupNode>(testGroupStats);
            node->children.swap(m_testCases);
            m_testGroups.push_back(node);
        }
        void testRunEnded(TestRunStats const& testRunStats) override {
            auto node = std::make_shared<TestRunNode>(testRunStats);
            node->children.swap(m_testGroups);
            m_testRuns.push_back(node);
            testRunEndedCumulative();
        }
        virtual void testRunEndedCumulative() = 0;

        void skipTest(TestCaseInfo const&) override {}

        IConfigPtr m_config;
        std::ostream& stream;
        std::vector<AssertionStats> m_assertions;
        std::vector<std::vector<std::shared_ptr<SectionNode»> m_sections;
        std::vector<std::shared_ptr<TestCaseNode» m_testCases;
        std::vector<std::shared_ptr<TestGroupNode» m_testGroups;

        std::vector<std::shared_ptr<TestRunNode» m_testRuns;

        std::shared_ptr<SectionNode> m_rootSection;
        std::shared_ptr<SectionNode> m_deepestSection;
        std::vector<std::shared_ptr<SectionNode» m_sectionStack;
        ReporterPreferences m_reporterPrefs;
    };

    template<char C>
    char const* getLineOfChars() {
        static char line[CATCH_CONFIG_CONSOLE_WIDTH] = {0};
        if( !*line ) {
            std::memset( line, C, CATCH_CONFIG_CONSOLE_WIDTH-1 );
            line[CATCH_CONFIG_CONSOLE_WIDTH-1] = 0;
        }
        return line;
    }

    struct TestEventListenerBase : StreamingReporterBase<TestEventListenerBase> {
        TestEventListenerBase( ReporterConfig const& _config );

        static std::set<Verbosity> getSupportedVerbosities();

        void assertionStarting(AssertionInfo const&) override;
        bool assertionEnded(AssertionStats const&) override;
    };

} // end namespace Catch

```

```
// end catch_reporter_bases.hpp
// start catch_console_colour.h

namespace Catch {

    struct Colour {
        enum Code {
            None = 0,

            White,
            Red,
            Green,
            Blue,
            Cyan,
            Yellow,
            Grey,

            Bright = 0x10,

            BrightRed = Bright | Red,
            BrightGreen = Bright | Green,
            LightGrey = Bright | Grey,
            BrightWhite = Bright | White,
            BrightYellow = Bright | Yellow,

            // By intention
            FileName = LightGrey,
            Warning = BrightYellow,
            ResultError = BrightRed,
            ResultSuccess = BrightGreen,
            ResultExpectedFailure = Warning,

            Error = BrightRed,
            Success = Green,

            OriginalExpression = Cyan,
            ReconstructedExpression = BrightYellow,

            SecondaryText = LightGrey,
            Headers = White
        };

        // Use constructed object for RAII guard
        Colour( Code _colourCode );
        Colour( Colour&& other ) noexcept;
        Colour& operator=( Colour&& other ) noexcept;
        ~Colour();

        // Use static method for one-shot changes
        static void use( Code _colourCode );

    private:
        bool m_moved = false;
    };

    std::ostream& operator « ( std::ostream& os, Colour const& );

} // end namespace Catch

// end catch_console_colour.h
// start catch_reporter_registrars.hpp


namespace Catch {

    template<typename T>
    class ReporterRegistrar {

        class ReporterFactory : public IReporterFactory {

            IStreamingReporterPtr create( ReporterConfig const& config ) const override {
                return std::unique_ptr<T>( new T( config ) );
            }

            std::string getDescription() const override {
                return T::getDescription();
            }
        };

    public:

        explicit ReporterRegistrar( std::string const& name ) {
            getMutableRegistryHub().registerReporter( name, std::make_shared<ReporterFactory>() );
        }
    };

    template<typename T>
    class ListenerRegistrar {

        class ListenerFactory : public IReporterFactory {

            IStreamingReporterPtr create( ReporterConfig const& config ) const override {
                return std::unique_ptr<T>( new T( config ) );
            }
            std::string getDescription() const override {
                return std::string();
            }
        };

    public:

        ListenerRegistrar() {
            getMutableRegistryHub().registerListener( std::make_shared<ListenerFactory>() );
        }
    };
}

#if !defined(CATCH_CONFIG_DISABLE)

#define CATCH_REGISTER_REPORTER( name, reporterType ) \
    CATCH_INTERNAL_START_WARNINGS_SUPPRESSION         \
    CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS          \
    namespace{ Catch::ReporterRegistrar<reporterType> catch_internal_RegistrarFor##reporterType( name
); } \
    CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

#define CATCH_REGISTER_LISTENER( listenerType ) \
    CATCH_INTERNAL_START_WARNINGS_SUPPRESSION   \
    CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS    \
    namespace{ Catch::ListenerRegistrar<listenerType> catch_internal_RegistrarFor##listenerType; } \
    CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION
#else // CATCH_CONFIG_DISABLE

#define CATCH_REGISTER_REPORTER(name, reporterType)
#define CATCH_REGISTER_LISTENER(listenerType)

#endif // CATCH_CONFIG_DISABLE

// end catch_reporter_registrars.hpp
// Allow users to base their work off existing reporters
// start catch_reporter_compact.h

namespace Catch {

    struct CompactReporter : StreamingReporterBase<CompactReporter> {

        using StreamingReporterBase::StreamingReporterBase;

        ~CompactReporter() override;

        static std::string getDescription();

        void noMatchingTestCases(std::string const& spec) override;

        void assertionStarting(AssertionInfo const&) override;

        bool assertionEnded(AssertionStats const& _assertionStats) override;

        void sectionEnded(SectionStats const& _sectionStats) override;

        void testRunEnded(TestRunStats const& _testRunStats) override;

    };

} // end namespace Catch

// end catch_reporter_compact.h
// start catch_reporter_console.h

#if defined(_MSC_VER)
#pragma warning(push)
#pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
                              // Note that 4062 (not all labels are handled
                              // and default is missing) is enabled
#endif

namespace Catch {
    // Fwd decls
    struct SummaryColumn;
    class TablePrinter;

    struct ConsoleReporter : StreamingReporterBase<ConsoleReporter> {
        std::unique_ptr<TablePrinter> m_tablePrinter;

        ConsoleReporter(ReporterConfig const& config);
        ~ConsoleReporter() override;
        static std::string getDescription();

        void noMatchingTestCases(std::string const& spec) override;

        void reportInvalidArguments(std::string const&arg) override;

        void assertionStarting(AssertionInfo const&) override;

        bool assertionEnded(AssertionStats const& _assertionStats) override;

        void sectionStarting(SectionInfo const& _sectionInfo) override;
        void sectionEnded(SectionStats const& _sectionStats) override;

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
        void benchmarkPreparing(std::string const& name) override;
        void benchmarkStarting(BenchmarkInfo const& info) override;
        void benchmarkEnded(BenchmarkStats<> const& stats) override;
        void benchmarkFailed(std::string const& error) override;
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

        void testCaseEnded(TestCaseStats const& _testCaseStats) override;
        void testGroupEnded(TestGroupStats const& _testGroupStats) override;
        void testRunEnded(TestRunStats const& _testRunStats) override;
        void testRunStarting(TestRunInfo const& _testRunInfo) override;
    private:

        void lazyPrint();

        void lazyPrintWithoutClosingBenchmarkTable();
        void lazyPrintRunInfo();
        void lazyPrintGroupInfo();
        void printTestCaseAndSectionHeader();

        void printClosedHeader(std::string const& _name);
        void printOpenHeader(std::string const& _name);

        // if string has a : in first line will set indent to follow it on
        // subsequent lines
        void printHeaderString(std::string const& _string, std::size_t indent = 0);

        void printTotals(Totals const& totals);
        void printSummaryRow(std::string const& label, std::vector<SummaryColumn> const& cols,
std::size_t row);

        void printTotalsDivider(Totals const& totals);
        void printSummaryDivider();
        void printTestFilters();

    private:
        bool m_headerPrinted = false;
    };

} // end namespace Catch

#if defined(_MSC_VER)
#pragma warning(pop)
#endif

// end catch_reporter_console.h
// start catch_reporter_junit.h

// start catch_xmlwriter.h

#include <vector>

namespace Catch {
    enum class XmlFormatting {
        None = 0x00,
        Indent = 0x01,
        Newline = 0x02,
    };

    XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs);
    XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs);

    class XmlEncode {
    public:
        enum ForWhat { ForTextNodes, ForAttributes };

        XmlEncode( std::string const& str, ForWhat forWhat = ForTextNodes );

        void encodeTo( std::ostream& os ) const;

        friend std::ostream& operator « ( std::ostream& os, XmlEncode const& xmlEncode );

    private:
        std::string m_str;
        ForWhat m_forWhat;
    };

    class XmlWriter {
    public:

        class ScopedElement {
        public:
            ScopedElement( XmlWriter* writer, XmlFormatting fmt );

            ScopedElement( ScopedElement&& other ) noexcept;
            ScopedElement& operator=( ScopedElement&& other ) noexcept;

            ~ScopedElement();

            ScopedElement& writeText( std::string const& text, XmlFormatting fmt =
XmlFormatting::Newline | XmlFormatting::Indent );

            template<typename T>
            ScopedElement& writeAttribute( std::string const& name, T const& attribute ) {
                m_writer->writeAttribute( name, attribute );
                return *this;
            }

        private:
            mutable XmlWriter* m_writer = nullptr;
            XmlFormatting m_fmt;
        };

        XmlWriter( std::ostream& os = Catch::cout() );
        ~XmlWriter();

        XmlWriter( XmlWriter const& ) = delete;
        XmlWriter& operator=( XmlWriter const& ) = delete;

        XmlWriter& startElement( std::string const& name, XmlFormatting fmt = XmlFormatting::Newline |
XmlFormatting::Indent);

        ScopedElement scopedElement( std::string const& name, XmlFormatting fmt =
XmlFormatting::Newline | XmlFormatting::Indent);

        XmlWriter& endElement(XmlFormatting fmt = XmlFormatting::Newline | XmlFormatting::Indent);

        XmlWriter& writeAttribute( std::string const& name, std::string const& attribute );

        XmlWriter& writeAttribute( std::string const& name, bool attribute );

        template<typename T>
        XmlWriter& writeAttribute( std::string const& name, T const& attribute ) {
            ReusableStringStream rss;
            rss « attribute;
            return writeAttribute( name, rss.str() );
        }

        XmlWriter& writeText( std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
XmlFormatting::Indent);

        XmlWriter& writeComment(std::string const& text, XmlFormatting fmt = XmlFormatting::Newline |
XmlFormatting::Indent);

        void writeStylesheetRef( std::string const& url );

        XmlWriter& writeBlankLine();

        void ensureTagClosed();

    private:

        void applyFormatting(XmlFormatting fmt);

        void writeDeclaration();

        void newlineIfNecessary();

        bool m_tagIsOpen = false;
        bool m_needsNewline = false;
        std::vector<std::string> m_tags;
        std::string m_indent;
        std::ostream& m_os;
    };

}

// end catch_xmlwriter.h
namespace Catch {

    class JunitReporter : public CumulativeReporterBase<JunitReporter> {
    public:
        JunitReporter(ReporterConfig const& _config);

        ~JunitReporter() override;

        static std::string getDescription();

        void noMatchingTestCases(std::string const& /*spec*/) override;

        void testRunStarting(TestRunInfo const& runInfo) override;

        void testGroupStarting(GroupInfo const& groupInfo) override;

        void testCaseStarting(TestCaseInfo const& testCaseInfo) override;
        bool assertionEnded(AssertionStats const& assertionStats) override;

        void testCaseEnded(TestCaseStats const& testCaseStats) override;

        void testGroupEnded(TestGroupStats const& testGroupStats) override;

        void testRunEndedCumulative() override;

        void writeGroup(TestGroupNode const& groupNode, double suiteTime);

        void writeTestCase(TestCaseNode const& testCaseNode);

        void writeSection( std::string const& className,
                           std::string const& rootName,
                           SectionNode const& sectionNode,
                           bool testOkToFail );

        void writeAssertions(SectionNode const& sectionNode);
        void writeAssertion(AssertionStats const& stats);

        XmlWriter xml;
        Timer suiteTimer;
        std::string stdOutForSuite;
        std::string stdErrForSuite;
        unsigned int unexpectedExceptions = 0;
        bool m_okToFail = false;
    };

} // end namespace Catch

// end catch_reporter_junit.h
// start catch_reporter_xml.h

namespace Catch {
    class XmlReporter : public StreamingReporterBase<XmlReporter> {
    public:
        XmlReporter(ReporterConfig const& _config);

        ~XmlReporter() override;

        static std::string getDescription();

        virtual std::string getStylesheetRef() const;

        void writeSourceInfo(SourceLineInfo const& sourceInfo);

    public: // StreamingReporterBase

        void noMatchingTestCases(std::string const& s) override;

        void testRunStarting(TestRunInfo const& testInfo) override;

        void testGroupStarting(GroupInfo const& groupInfo) override;

        void testCaseStarting(TestCaseInfo const& testInfo) override;

        void sectionStarting(SectionInfo const& sectionInfo) override;

        void assertionStarting(AssertionInfo const&) override;

        bool assertionEnded(AssertionStats const& assertionStats) override;

        void sectionEnded(SectionStats const& sectionStats) override;

        void testCaseEnded(TestCaseStats const& testCaseStats) override;

        void testGroupEnded(TestGroupStats const& testGroupStats) override;

        void testRunEnded(TestRunStats const& testRunStats) override;

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
```

```
        void benchmarkPreparing(std::string const& name) override;
        void benchmarkStarting(BenchmarkInfo const&) override;
        void benchmarkEnded(BenchmarkStats<> const&) override;
        void benchmarkFailed(std::string const&) override;
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

    private:
        Timer m_testCaseTimer;
        XmlWriter m_xml;
        int m_sectionDepth = 0;
    };

} // end namespace Catch

// end catch_reporter_xml.h

// end catch_external_interfaces.h
#endif

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
// start catch_benchmarking_all.hpp

// A proxy header that includes all of the benchmarking headers to allow
// concise include of the benchmarking features. You should prefer the
// individual includes in standard use.

// start catch_benchmark.hpp

 // Benchmark

// start catch_chronometer.hpp

// User-facing chronometer


// start catch_clock.hpp

// Clocks


#include <chrono>
#include <ratio>

namespace Catch {
    namespace Benchmark {
        template <typename Clock>
        using ClockDuration = typename Clock::duration;
        template <typename Clock>
        using FloatDuration = std::chrono::duration<double, typename Clock::period>;

        template <typename Clock>
        using TimePoint = typename Clock::time_point;

        using default_clock = std::chrono::steady_clock;

        template <typename Clock>
        struct now {
            TimePoint<Clock> operator()() const {
                return Clock::now();
            }
        };

        using fp_seconds = std::chrono::duration<double, std::ratio<1»;
    } // namespace Benchmark
} // namespace Catch

// end catch_clock.hpp
// start catch_optimizer.hpp

 // Hinting the optimizer


#if defined(_MSC_VER)
#   include <atomic> // atomic_thread_fence
#endif

namespace Catch {
    namespace Benchmark {
#if defined(__GNUC__) || defined(__clang__)
        template <typename T>
        inline void keep_memory(T* p) {
            asm volatile("" : : "g"(p) : "memory");
        }
        inline void keep_memory() {
            asm volatile("" : : : "memory");
        }

        namespace Detail {
            inline void optimizer_barrier() { keep_memory(); }
        } // namespace Detail
#elif defined(_MSC_VER)

#pragma optimize("", off)
        template <typename T>
        inline void keep_memory(T* p) {
            // thanks @milleniumbug
            *reinterpret_cast<char volatile*>(p) = *reinterpret_cast<char const volatile*>(p);
        }
        // TODO equivalent keep_memory()
#pragma optimize("", on)

        namespace Detail {
            inline void optimizer_barrier() {
                std::atomic_thread_fence(std::memory_order_seq_cst);
            }
        } // namespace Detail

#endif

        template <typename T>
        inline void deoptimize_value(T&& x) {
            keep_memory(&x);
        }

        template <typename Fn, typename... Args>
        inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<!std::is_same<void, decltype(fn(args...))>::value>::type {
            deoptimize_value(std::forward<Fn>(fn) (std::forward<Args...>(args...)));
        }

        template <typename Fn, typename... Args>
        inline auto invoke_deoptimized(Fn&& fn, Args&&... args) -> typename
std::enable_if<std::is_same<void, decltype(fn(args...))>::value>::type {
            std::forward<Fn>(fn) (std::forward<Args...>(args...));
        }
    } // namespace Benchmark
} // namespace Catch

// end catch_optimizer.hpp
// start catch_complete_invoke.hpp

// Invoke with a special case for void


#include <type_traits>
#include <utility>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename T>
            struct CompleteType { using type = T; };
            template <>
            struct CompleteType<void> { struct type {}; };

            template <typename T>
            using CompleteType_t = typename CompleteType<T>::type;

            template <typename Result>
            struct CompleteInvoker {
                template <typename Fun, typename... Args>
                static Result invoke(Fun&& fun, Args&&... args) {
                    return std::forward<Fun>(fun)(std::forward<Args>(args)...);
                }
            };
            template <>
            struct CompleteInvoker<void> {
                template <typename Fun, typename... Args>
                static CompleteType_t<void> invoke(Fun&& fun, Args&&... args) {
                    std::forward<Fun>(fun)(std::forward<Args>(args)...);
                    return {};
                }
            };

            // invoke and not return void :(
            template <typename Fun, typename... Args>
            CompleteType_t<FunctionReturnType<Fun, Args...» complete_invoke(Fun&& fun, Args&&... args)
{
                return CompleteInvoker<FunctionReturnType<Fun,
Args...»::invoke(std::forward<Fun>(fun), std::forward<Args>(args)...);
            }

            const std::string benchmarkErrorMsg = "a benchmark failed to run successfully";
        } // namespace Detail

        template <typename Fun>
        Detail::CompleteType_t<FunctionReturnType<Fun>> user_code(Fun&& fun) {
            CATCH_TRY{
                return Detail::complete_invoke(std::forward<Fun>(fun));
            } CATCH_CATCH_ALL{
                getResultCapture().benchmarkFailed(translateActiveException());
                CATCH_RUNTIME_ERROR(Detail::benchmarkErrorMsg);
            }
        }
    } // namespace Benchmark
} // namespace Catch

// end catch_complete_invoke.hpp
namespace Catch {
    namespace Benchmark {
        namespace Detail {
            struct ChronometerConcept {
                virtual void start() = 0;
                virtual void finish() = 0;
                virtual ~ChronometerConcept() = default;
            };
            template <typename Clock>
            struct ChronometerModel final : public ChronometerConcept {
                void start() override { started = Clock::now(); }
                void finish() override { finished = Clock::now(); }

                ClockDuration<Clock> elapsed() const { return finished - started; }

                TimePoint<Clock> started;
                TimePoint<Clock> finished;
            };
        } // namespace Detail

        struct Chronometer {
        public:
            template <typename Fun>
            void measure(Fun&& fun) { measure(std::forward<Fun>(fun), is_callable<Fun(int)>()); }

            int runs() const { return k; }

            Chronometer(Detail::ChronometerConcept& meter, int k)
                : impl(&meter)
                , k(k) {}

        private:
            template <typename Fun>
            void measure(Fun&& fun, std::false_type) {
                measure([&fun](int) { return fun(); }, std::true_type());
            }

            template <typename Fun>
            void measure(Fun&& fun, std::true_type) {
                Detail::optimizer_barrier();
                impl->start();
                for (int i = 0; i < k; ++i) invoke_deoptimized(fun, i);
                impl->finish();
                Detail::optimizer_barrier();
            }

            Detail::ChronometerConcept* impl;
            int k;
        };
    } // namespace Benchmark
} // namespace Catch

// end catch_chronometer.hpp
// start catch_environment.hpp

// Environment information


namespace Catch {
    namespace Benchmark {
        template <typename Duration>
        struct EnvironmentEstimate {
            Duration mean;
            OutlierClassification outliers;

            template <typename Duration2>
            operator EnvironmentEstimate<Duration2>() const {
                return { mean, outliers };
            }
        };
        template <typename Clock>
        struct Environment {
            using clock_type = Clock;
            EnvironmentEstimate<FloatDuration<Clock>> clock_resolution;
            EnvironmentEstimate<FloatDuration<Clock>> clock_cost;
        };
    } // namespace Benchmark
} // namespace Catch

// end catch_environment.hpp
// start catch_execution_plan.hpp

 // Execution plan


// start catch_benchmark_function.hpp

 // Dumb std::function implementation for consistent call overhead


#include <cassert>
#include <type_traits>
#include <utility>
#include <memory>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename T>
            using Decay = typename std::decay<T>::type;
            template <typename T, typename U>
            struct is_related
                : std::is_same<Decay<T>, Decay<U>> {};

            struct BenchmarkFunction {
            private:
                struct callable {
                    virtual void call(Chronometer meter) const = 0;
                    virtual callable* clone() const = 0;
                    virtual ~callable() = default;
                };
                template <typename Fun>
                struct model : public callable {
                    model(Fun&& fun) : fun(std::move(fun)) {}
                    model(Fun const& fun) : fun(fun) {}

                    model<Fun>* clone() const override { return new model<Fun>(*this); }

                    void call(Chronometer meter) const override {
                        call(meter, is_callable<Fun(Chronometer)>());
                    }
                    void call(Chronometer meter, std::true_type) const {
                        fun(meter);
                    }
                    void call(Chronometer meter, std::false_type) const {
                        meter.measure(fun);
                    }

                    Fun fun;
                };

                struct do_nothing { void operator()() const {} };

                template <typename T>
                BenchmarkFunction(model<T>* c) : f(c) {}

            public:
                BenchmarkFunction()
                    : f(new model<do_nothing>{ {} }) {}

                template <typename Fun,
                    typename std::enable_if<!is_related<Fun, BenchmarkFunction>::value, int>::type =
0>
                    BenchmarkFunction(Fun&& fun)
                    : f(new model<typename std::decay<Fun>::type>(std::forward<Fun>(fun))) {}

                BenchmarkFunction(BenchmarkFunction&& that)
                    : f(std::move(that.f)) {}

                BenchmarkFunction(BenchmarkFunction const& that)
                    : f(that.f->clone()) {}

                BenchmarkFunction& operator=(BenchmarkFunction&& that) {
                    f = std::move(that.f);
                    return *this;
                }

                BenchmarkFunction& operator=(BenchmarkFunction const& that) {
                    f.reset(that.f->clone());
                    return *this;
                }

                void operator()(Chronometer meter) const { f->call(meter); }

            private:
                std::unique_ptr<callable> f;
            };
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

// end catch_benchmark_function.hpp
// start catch_repeat.hpp

// repeat algorithm


#include <type_traits>
#include <utility>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename Fun>
            struct repeater {
                void operator()(int k) const {
                    for (int i = 0; i < k; ++i) {
                        fun();
                    }
                }
                Fun fun;
            };
            template <typename Fun>
            repeater<typename std::decay<Fun>::type> repeat(Fun&& fun) {
                return { std::forward<Fun>(fun) };
            }
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

// end catch_repeat.hpp
// start catch_run_for_at_least.hpp

// Run a function for a minimum amount of time


// start catch_measure.hpp

// Measure


// start catch_timing.hpp

// Timing


#include <tuple>
#include <type_traits>

namespace Catch {
    namespace Benchmark {
        template <typename Duration, typename Result>
        struct Timing {
            Duration elapsed;
            Result result;
            int iterations;
        };
        template <typename Clock, typename Func, typename... Args>
        using TimingOf = Timing<ClockDuration<Clock>, Detail::CompleteType_t<FunctionReturnType<Func,
Args...»>;
    } // namespace Benchmark
} // namespace Catch

// end catch_timing.hpp
#include <utility>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename Clock, typename Fun, typename... Args>
            TimingOf<Clock, Fun, Args...> measure(Fun&& fun, Args&&... args) {
                auto start = Clock::now();
                auto&& r = Detail::complete_invoke(fun, std::forward<Args>(args)...);
                auto end = Clock::now();
                auto delta = end - start;
                return { delta, std::forward<decltype(r)>(r), 1 };
            }
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

// end catch_measure.hpp
#include <utility>
#include <type_traits>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename Clock, typename Fun>
            TimingOf<Clock, Fun, int> measure_one(Fun&& fun, int iters, std::false_type) {
                return Detail::measure<Clock>(fun, iters);
            }
            template <typename Clock, typename Fun>
            TimingOf<Clock, Fun, Chronometer> measure_one(Fun&& fun, int iters, std::true_type) {
                Detail::ChronometerModel<Clock> meter;
                auto&& result = Detail::complete_invoke(fun, Chronometer(meter, iters));

                return { meter.elapsed(), std::move(result), iters };
            }

            template <typename Clock, typename Fun>
            using run_for_at_least_argument_t = typename
std::conditional<is_callable<Fun(Chronometer)>::value, Chronometer, int>::type;

            struct optimized_away_error : std::exception {
                const char* what() const noexcept override {
                    return "could not measure benchmark, maybe it was optimized away";
                }
            };

            template <typename Clock, typename Fun>
            TimingOf<Clock, Fun, run_for_at_least_argument_t<Clock, Fun»
run_for_at_least(ClockDuration<Clock> how_long, int seed, Fun&& fun) {
                auto iters = seed;
                while (iters < (1 « 30)) {
                    auto&& Timing = measure_one<Clock>(fun, iters, is_callable<Fun(Chronometer)>());

                    if (Timing.elapsed >= how_long) {
                        return { Timing.elapsed, std::move(Timing.result), iters };
                    }
                    iters *= 2;
                }
                Catch::throw_exception(optimized_away_error{});
            }
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

// end catch_run_for_at_least.hpp
#include <algorithm>
#include <iterator>

namespace Catch {
    namespace Benchmark {
        template <typename Duration>
        struct ExecutionPlan {
            int iterations_per_sample;
            Duration estimated_duration;
            Detail::BenchmarkFunction benchmark;
            Duration warmup_time;
            int warmup_iterations;

            template <typename Duration2>
            operator ExecutionPlan<Duration2>() const {
                return { iterations_per_sample, estimated_duration, benchmark, warmup_time,
warmup_iterations };
            }

            template <typename Clock>
            std::vector<FloatDuration<Clock» run(const IConfig &cfg, Environment<FloatDuration<Clock»
env) const {
                // warmup a bit

Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock»(warmup_time),
warmup_iterations, Detail::repeat(now<Clock>{}));

                std::vector<FloatDuration<Clock» times;
                times.reserve(cfg.benchmarkSamples());
                std::generate_n(std::back_inserter(times), cfg.benchmarkSamples(), [this, env] {
                    Detail::ChronometerModel<Clock> model;
                    this->benchmark(Chronometer(model, iterations_per_sample));
                    auto sample_time = model.elapsed() - env.clock_cost.mean;
                    if (sample_time < FloatDuration<Clock>::zero()) sample_time =
FloatDuration<Clock>::zero();
```

```
                    return sample_time / iterations_per_sample;
                });
                return times;
            }
        };
    } // namespace Benchmark
} // namespace Catch

// end catch_execution_plan.hpp
// start catch_estimate_clock.hpp

 // Environment measurement


// start catch_stats.hpp

// Statistical analysis tools


#include <algorithm>
#include <functional>
#include <vector>
#include <iterator>
#include <numeric>
#include <tuple>
#include <cmath>
#include <utility>
#include <cstddef>
#include <random>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            using sample = std::vector<double>;

            double weighted_average_quantile(int k, int q, std::vector<double>::iterator first,
std::vector<double>::iterator last);

            template <typename Iterator>
            OutlierClassification classify_outliers(Iterator first, Iterator last) {
                std::vector<double> copy(first, last);

                auto q1 = weighted_average_quantile(1, 4, copy.begin(), copy.end());
                auto q3 = weighted_average_quantile(3, 4, copy.begin(), copy.end());
                auto iqr = q3 - q1;
                auto los = q1 - (iqr * 3.);
                auto lom = q1 - (iqr * 1.5);
                auto him = q3 + (iqr * 1.5);
                auto his = q3 + (iqr * 3.);

                OutlierClassification o;
                for (; first != last; ++first) {
                    auto&& t = *first;
                    if (t < los) ++o.low_severe;
                    else if (t < lom) ++o.low_mild;
                    else if (t > his) ++o.high_severe;
                    else if (t > him) ++o.high_mild;
                    ++o.samples_seen;
                }
                return o;
            }

            template <typename Iterator>
            double mean(Iterator first, Iterator last) {
                auto count = last - first;
                double sum = std::accumulate(first, last, 0.);
                return sum / count;
            }

            template <typename URng, typename Iterator, typename Estimator>
            sample resample(URng& rng, int resamples, Iterator first, Iterator last, Estimator&
estimator) {
                auto n = last - first;
                std::uniform_int_distribution<decltype(n)> dist(0, n - 1);

                sample out;
                out.reserve(resamples);
                std::generate_n(std::back_inserter(out), resamples, [n, first, &estimator, &dist,
&rng] {
                    std::vector<double> resampled;
                    resampled.reserve(n);
                    std::generate_n(std::back_inserter(resampled), n, [first, &dist, &rng] { return
first[dist(rng)]; });
                    return estimator(resampled.begin(), resampled.end());
                });
                std::sort(out.begin(), out.end());
                return out;
            }

            template <typename Estimator, typename Iterator>
            sample jackknife(Estimator&& estimator, Iterator first, Iterator last) {
                auto n = last - first;
                auto second = std::next(first);
                sample results;
                results.reserve(n);

                for (auto it = first; it != last; ++it) {
                    std::iter_swap(it, first);
                    results.push_back(estimator(second, last));
                }

                return results;
            }

            inline double normal_cdf(double x) {
                return std::erfc(-x / std::sqrt(2.0)) / 2.0;
            }

            double erfc_inv(double x);

            double normal_quantile(double p);

            template <typename Iterator, typename Estimator>
            Estimate<double> bootstrap(double confidence_level, Iterator first, Iterator last, sample const& resample, Estimator&& estimator) {
                auto n_samples = last - first;

                double point = estimator(first, last);
                // Degenerate case with a single sample
                if (n_samples == 1) return { point, point, point, confidence_level };

                sample jack = jackknife(estimator, first, last);
                double jack_mean = mean(jack.begin(), jack.end());
                double sum_squares, sum_cubes;
                std::tie(sum_squares, sum_cubes) = std::accumulate(jack.begin(), jack.end(), std::make_pair(0., 0.), [jack_mean](std::pair<double, double> sqcb, double x) -> std::pair<double, double> {
                    auto d = jack_mean - x;
                    auto d2 = d * d;
                    auto d3 = d2 * d;
                    return { sqcb.first + d2, sqcb.second + d3 };
                });

                double accel = sum_cubes / (6 * std::pow(sum_squares, 1.5));
                int n = static_cast<int>(resample.size());
                double prob_n = std::count_if(resample.begin(), resample.end(), [point](double x) { return x < point; }) / (double)n;
                // degenerate case with uniform samples
                if (prob_n == 0) return { point, point, point, confidence_level };

                double bias = normal_quantile(prob_n);
                double z1 = normal_quantile((1. - confidence_level) / 2.);

                auto cumn = [n](double x) -> int {
                    return std::lround(normal_cdf(x) * n); };
                auto a = [bias, accel](double b) { return bias + b / (1. - accel * b); };
                double b1 = bias + z1;
                double b2 = bias - z1;
                double a1 = a(b1);
                double a2 = a(b2);
                auto lo = (std::max)(cumn(a1), 0);
                auto hi = (std::min)(cumn(a2), n - 1);

                return { point, resample[lo], resample[hi], confidence_level };
            }

            double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n);

            struct bootstrap_analysis {
                Estimate<double> mean;
                Estimate<double> standard_deviation;
                double outlier_variance;
            };

            bootstrap_analysis analyse_samples(double confidence_level, int n_resamples, std::vector<double>::iterator first, std::vector<double>::iterator last);
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

// end catch_stats.hpp
#include <algorithm>
#include <iterator>
#include <tuple>
```

```
#include <vector>
#include <cmath>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename Clock>
            std::vector<double> resolution(int k) {
                std::vector<TimePoint<Clock>> times;
                times.reserve(k + 1);
                std::generate_n(std::back_inserter(times), k + 1, now<Clock>{});

                std::vector<double> deltas;
                deltas.reserve(k);
                std::transform(std::next(times.begin()), times.end(), times.begin(),
                    std::back_inserter(deltas),
                    [](TimePoint<Clock> a, TimePoint<Clock> b) { return static_cast<double>((a -
b).count()); });

                return deltas;
            }

            const auto warmup_iterations = 10000;
            const auto warmup_time = std::chrono::milliseconds(100);
            const auto minimum_ticks = 1000;
            const auto warmup_seed = 10000;
            const auto clock_resolution_estimation_time = std::chrono::milliseconds(500);
            const auto clock_cost_estimation_time_limit = std::chrono::seconds(1);
            const auto clock_cost_estimation_tick_limit = 100000;
            const auto clock_cost_estimation_time = std::chrono::milliseconds(10);
            const auto clock_cost_estimation_iterations = 10000;

            template <typename Clock>
            int warmup() {
                return
run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(warmup_time), warmup_seed,
&resolution<Clock>)
                    .iterations;
            }
            template <typename Clock>
            EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_resolution(int iterations) {
                auto r =
run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_resolution_estimation_time),
iterations, &resolution<Clock>)
                    .result;
                return {
                    FloatDuration<Clock>(mean(r.begin(), r.end())),
                    classify_outliers(r.begin(), r.end()),
                };
            }
            template <typename Clock>
            EnvironmentEstimate<FloatDuration<Clock>> estimate_clock_cost(FloatDuration<Clock>
resolution) {
                auto time_limit = (std::min)(
                    resolution * clock_cost_estimation_tick_limit,
                    FloatDuration<Clock>(clock_cost_estimation_time_limit));
                auto time_clock = [](int k) {
                    return Detail::measure<Clock>([k] {
                        for (int i = 0; i < k; ++i) {
                            volatile auto ignored = Clock::now();
                            (void)ignored;
                        }
                    }).elapsed;
                };
                time_clock(1);
                int iters = clock_cost_estimation_iterations;
                auto&& r =
run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(clock_cost_estimation_time),
iters, time_clock);
                std::vector<double> times;
                int nsamples = static_cast<int>(std::ceil(time_limit / r.elapsed));
                times.reserve(nsamples);
                std::generate_n(std::back_inserter(times), nsamples, [time_clock, &r] {
                    return static_cast<double>((time_clock(r.iterations) / r.iterations).count());
                });
                return {
                    FloatDuration<Clock>(mean(times.begin(), times.end())),
                    classify_outliers(times.begin(), times.end()),
                };
            }

            template <typename Clock>
            Environment<FloatDuration<Clock>> measure_environment() {
                static Environment<FloatDuration<Clock>>* env = nullptr;
                if (env) {
                    return *env;
                }

                auto iters = Detail::warmup<Clock>();
                auto resolution = Detail::estimate_clock_resolution<Clock>(iters);
                auto cost = Detail::estimate_clock_cost<Clock>(resolution.mean);

                env = new Environment<FloatDuration<Clock>>{ resolution, cost };
                return *env;
            }
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

// end catch_estimate_clock.hpp
// start catch_analyse.hpp

 // Run and analyse one benchmark


// start catch_sample_analysis.hpp

// Benchmark results


#include <algorithm>
#include <vector>
#include <string>
#include <iterator>

namespace Catch {
    namespace Benchmark {
        template <typename Duration>
        struct SampleAnalysis {
            std::vector<Duration> samples;
            Estimate<Duration> mean;
            Estimate<Duration> standard_deviation;
            OutlierClassification outliers;
            double outlier_variance;

            template <typename Duration2>
            operator SampleAnalysis<Duration2>() const {
                std::vector<Duration2> samples2;
                samples2.reserve(samples.size());
                std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
[](Duration d) { return Duration2(d); });
                return {
                    std::move(samples2),
                    mean,
                    standard_deviation,
                    outliers,
                    outlier_variance,
                };
            }
        };
    } // namespace Benchmark
} // namespace Catch

// end catch_sample_analysis.hpp
#include <algorithm>
#include <iterator>
#include <vector>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename Duration, typename Iterator>
            SampleAnalysis<Duration> analyse(const IConfig &cfg, Environment<Duration>, Iterator
first, Iterator last) {
                if (!cfg.benchmarkNoAnalysis()) {
                    std::vector<double> samples;
                    samples.reserve(last - first);
                    std::transform(first, last, std::back_inserter(samples), [](Duration d) { return
d.count(); });

                    auto analysis =
Catch::Benchmark::Detail::analyse_samples(cfg.benchmarkConfidenceInterval(), cfg.benchmarkResamples(),
samples.begin(), samples.end());
                    auto outliers = Catch::Benchmark::Detail::classify_outliers(samples.begin(),
samples.end());

                    auto wrap_estimate = [](Estimate<double> e) {
                        return Estimate<Duration> {
                            Duration(e.point),
                                Duration(e.lower_bound),
                                Duration(e.upper_bound),
                                e.confidence_interval,
                        };
                    };
                    std::vector<Duration> samples2;
                    samples2.reserve(samples.size());
                    std::transform(samples.begin(), samples.end(), std::back_inserter(samples2),
[](double d) { return Duration(d); });
                    return {
                        std::move(samples2),
                        wrap_estimate(analysis.mean),
                        wrap_estimate(analysis.standard_deviation),
                        outliers,
                        analysis.outlier_variance,
                    };
                } else {
                    std::vector<Duration> samples;
                    samples.reserve(last - first);

                    Duration mean = Duration(0);
                    int i = 0;
                    for (auto it = first; it < last; ++it, ++i) {
                        samples.push_back(Duration(*it));
                        mean += Duration(*it);
                    }
                    mean /= i;

                    return {
                        std::move(samples),
                        Estimate<Duration>{mean, mean, mean, 0.0},
                        Estimate<Duration>{Duration(0), Duration(0), Duration(0), 0.0},
                        OutlierClassification{},
                        0.0
                    };
                }
            }
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

// end catch_analyse.hpp
#include <algorithm>
#include <functional>
#include <string>
#include <vector>
#include <cmath>

namespace Catch {
    namespace Benchmark {
        struct Benchmark {
            Benchmark(std::string &&name)
                : name(std::move(name)) {}

            template <class FUN>
            Benchmark(std::string &&name, FUN &&func)
                : fun(std::move(func)), name(std::move(name)) {}

            template <typename Clock>
            ExecutionPlan<FloatDuration<Clock>> prepare(const IConfig &cfg,
Environment<FloatDuration<Clock>> env) const {
                auto min_time = env.clock_resolution.mean * Detail::minimum_ticks;
                auto run_time = std::max(min_time,
std::chrono::duration_cast<decltype(min_time)>(cfg.benchmarkWarmupTime()));
                auto&& test =
Detail::run_for_at_least<Clock>(std::chrono::duration_cast<ClockDuration<Clock>>(run_time), 1, fun);
                int new_iters = static_cast<int>(std::ceil(min_time * test.iterations /
test.elapsed));
                return { new_iters, test.elapsed / test.iterations * new_iters *
cfg.benchmarkSamples(), fun,
std::chrono::duration_cast<FloatDuration<Clock>>(cfg.benchmarkWarmupTime()), Detail::warmup_iterations
};
            }

            template <typename Clock = default_clock>
            void run() {
                IConfigPtr cfg = getCurrentContext().getConfig();

                auto env = Detail::measure_environment<Clock>();

                getResultCapture().benchmarkPreparing(name);
                CATCH_TRY{
                    auto plan = user_code([&] {
                        return prepare<Clock>(*cfg, env);
                    });

                    BenchmarkInfo info {
                        name,
                        plan.estimated_duration.count(),
                        plan.iterations_per_sample,
                        cfg->benchmarkSamples(),
                        cfg->benchmarkResamples(),
                        env.clock_resolution.mean.count(),
                        env.clock_cost.mean.count()
                    };

                    getResultCapture().benchmarkStarting(info);

                    auto samples = user_code([&] {
                        return plan.template run<Clock>(*cfg, env);
                    });

                    auto analysis = Detail::analyse(*cfg, env, samples.begin(), samples.end());
                    BenchmarkStats<FloatDuration<Clock>> stats{ info, analysis.samples, analysis.mean,
analysis.standard_deviation, analysis.outliers, analysis.outlier_variance };
                    getResultCapture().benchmarkEnded(stats);

                } CATCH_CATCH_ALL{
                    if (translateActiveException() != Detail::benchmarkErrorMsg) // benchmark errors
have been reported, otherwise rethrow.
                        std::rethrow_exception(std::current_exception());
                }
            }

            // sets lambda to be used in fun *and* executes benchmark!
            template <typename Fun,
                typename std::enable_if<!Detail::is_related<Fun, Benchmark>::value, int>::type = 0>
                Benchmark & operator=(Fun func) {
                fun = Detail::BenchmarkFunction(func);
                run();
                return *this;
            }

            explicit operator bool() {
                return true;
            }

        private:
            Detail::BenchmarkFunction fun;
            std::string name;
        };
    }
} // namespace Catch

#define INTERNAL_CATCH_GET_1_ARG(arg1, arg2, ...) arg1
#define INTERNAL_CATCH_GET_2_ARG(arg1, arg2, ...) arg2

#define INTERNAL_CATCH_BENCHMARK(BenchmarkName, name, benchmarkIndex)\
    if( Catch::Benchmark::Benchmark BenchmarkName{name} ) \
        BenchmarkName = [&](int benchmarkIndex)

#define INTERNAL_CATCH_BENCHMARK_ADVANCED(BenchmarkName, name)\
    if( Catch::Benchmark::Benchmark BenchmarkName{name} ) \
        BenchmarkName = [&]

// end catch_benchmark.hpp
// start catch_constructor.hpp

// Constructor and destructor helpers


#include <type_traits>

namespace Catch {
    namespace Benchmark {
        namespace Detail {
            template <typename T, bool Destruct>
            struct ObjectStorage
            {
                ObjectStorage() : data() {}

                ObjectStorage(const ObjectStorage& other)
                {
                    new(&data) T(other.stored_object());
                }

                ObjectStorage(ObjectStorage&& other)
                {
                    new(&data) T(std::move(other.stored_object()));
                }

                ~ObjectStorage() { destruct_on_exit<T>(); }

                template <typename... Args>
                void construct(Args&&... args)
                {
                    new (&data) T(std::forward<Args>(args)...);
                }

                template <bool AllowManualDestruction = !Destruct>
                typename std::enable_if<AllowManualDestruction>::type destruct()
                {
                    stored_object().~T();
                }

            private:
                // If this is a constructor benchmark, destruct the underlying object
                template <typename U>
                void destruct_on_exit(typename std::enable_if<Destruct, U>::type* = 0) {
destruct<true>(); }
                // Otherwise, don't
                template <typename U>
                void destruct_on_exit(typename std::enable_if<!Destruct, U>::type* = 0) { }

                T& stored_object() {
                    return *static_cast<T*>(static_cast<void*>(&data));
                }

                T const& stored_object() const {
                    return *static_cast<T*>(static_cast<void*>(&data));
                }

                struct { alignas(T) unsigned char data[sizeof(T)]; }  data;
            };
        }

        template <typename T>
        using storage_for = Detail::ObjectStorage<T, true>;

        template <typename T>
        using destructable_object = Detail::ObjectStorage<T, false>;
    }
}

// end catch_constructor.hpp
// end catch_benchmarking_all.hpp
#endif

#endif // ! CATCH_CONFIG_IMPL_ONLY

#ifdef CATCH_IMPL
// start catch_impl.hpp

#ifdef __clang__
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wweak-vtables"
#endif

// Keep these here for external reporters
// start catch_test_case_tracker.h

#include <string>
#include <vector>
#include <memory>

namespace Catch {
namespace TestCaseTracking {

    struct NameAndLocation {
        std::string name;
        SourceLineInfo location;

        NameAndLocation( std::string const& _name, SourceLineInfo const& _location );
        friend bool operator==(NameAndLocation const& lhs, NameAndLocation const& rhs) {
            return lhs.name == rhs.name
                && lhs.location == rhs.location;
        }
    };

    class ITracker;

    using ITrackerPtr = std::shared_ptr<ITracker>;

    class  ITracker {
        NameAndLocation m_nameAndLocation;

    public:
        ITracker(NameAndLocation const& nameAndLoc) :
            m_nameAndLocation(nameAndLoc)
        {}

        // static queries
        NameAndLocation const& nameAndLocation() const {
            return m_nameAndLocation;
        }

        virtual ~ITracker();

        // dynamic queries
        virtual bool isComplete() const = 0; // Successfully completed or failed
        virtual bool isSuccessfullyCompleted() const = 0;
        virtual bool isOpen() const = 0; // Started but not complete
        virtual bool hasChildren() const = 0;
        virtual bool hasStarted() const = 0;

        virtual ITracker& parent() = 0;

        // actions
        virtual void close() = 0; // Successfully complete
        virtual void fail() = 0;
        virtual void markAsNeedingAnotherRun() = 0;

        virtual void addChild( ITrackerPtr const& child ) = 0;
        virtual ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) = 0;
        virtual void openChild() = 0;

        // Debug/ checking
        virtual bool isSectionTracker() const = 0;
        virtual bool isGeneratorTracker() const = 0;
    };

    class TrackerContext {

        enum RunState {
            NotStarted,
            Executing,
            CompletedCycle
        };

        ITrackerPtr m_rootTracker;
        ITracker* m_currentTracker = nullptr;
        RunState m_runState = NotStarted;

    public:

        ITracker& startRun();
        void endRun();

        void startCycle();
        void completeCycle();

        bool completedCycle() const;
        ITracker& currentTracker();
        void setCurrentTracker( ITracker* tracker );
    };

    class TrackerBase : public ITracker {
    protected:
        enum CycleState {
            NotStarted,
            Executing,
            ExecutingChildren,
            NeedsAnotherRun,
            CompletedSuccessfully,
            Failed
        };

        using Children = std::vector<ITrackerPtr>;
        TrackerContext& m_ctx;
        ITracker* m_parent;
        Children m_children;
        CycleState m_runState = NotStarted;

    public:
        TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent );

        bool isComplete() const override;
        bool isSuccessfullyCompleted() const override;
        bool isOpen() const override;
        bool hasChildren() const override;
        bool hasStarted() const override {
            return m_runState != NotStarted;
        }

        void addChild( ITrackerPtr const& child ) override;

        ITrackerPtr findChild( NameAndLocation const& nameAndLocation ) override;
        ITracker& parent() override;

        void openChild() override;

        bool isSectionTracker() const override;
        bool isGeneratorTracker() const override;

        void open();

        void close() override;
        void fail() override;
        void markAsNeedingAnotherRun() override;

    private:
        void moveToParent();
        void moveToThis();
    };

    class SectionTracker : public TrackerBase {
        std::vector<std::string> m_filters;
        std::string m_trimmed_name;
    public:
        SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent
);

        bool isSectionTracker() const override;

        bool isComplete() const override;

        static SectionTracker& acquire( TrackerContext& ctx, NameAndLocation const& nameAndLocation );

        void tryOpen();

        void addInitialFilters( std::vector<std::string> const& filters );
        void addNextFilters( std::vector<std::string> const& filters );
        std::vector<std::string> const& getFilters() const;
        std::string const& trimmedName() const;
    };

} // namespace TestCaseTracking

using TestCaseTracking::ITracker;
using TestCaseTracking::TrackerContext;
using TestCaseTracking::SectionTracker;

} // namespace Catch

// end catch_test_case_tracker.h

// start catch_leak_detector.h

namespace Catch {

    struct LeakDetector {
        LeakDetector();
        ~LeakDetector();
    };

}
// end catch_leak_detector.h
// Cpp files will be included in the single-header file here
// start catch_stats.cpp

// Statistical analysis tools

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)

#include <cassert>
#include <random>

#if defined(CATCH_CONFIG_USE_ASYNC)
#include <future>
#endif

namespace {
    double erf_inv(double x) {
        // Code accompanying the article "Approximating the erfinv function" in GPU Computing Gems,
Volume 2
        double w, p;

        w = -log((1.0 - x) * (1.0 + x));

        if (w < 6.250000) {
            w = w - 3.125000;
            p = -3.6444120640178196996e-21;
            p = -1.685059138182016589e-19 + p * w;
            p = 1.2858480715256400167e-18 + p * w;
            p = 1.115787767802518096e-17 + p * w;
            p = -1.333171662854620906e-16 + p * w;
            p = 2.0972767875968561637e-17 + p * w;
            p = 6.6376381343583238325e-15 + p * w;
            p = -4.0545662729752068639e-14 + p * w;
            p = -8.1519341976054721522e-14 + p * w;
            p = 2.6335093153082322977e-12 + p * w;
            p = -1.2975133253453532498e-11 + p * w;
            p = -5.4154120542946279317e-11 + p * w;
            p = 1.051212273321532285e-09 + p * w;
            p = -4.1126339803469836976e-09 + p * w;
            p = -2.9070369957882005086e-08 + p * w;
            p = 4.2347877827932403518e-07 + p * w;
            p = -1.3654692000834678645e-06 + p * w;
            p = -1.3882523362786468719e-05 + p * w;
            p = 0.0001867342080340571352 + p * w;
            p = -0.00074070253416626697512 + p * w;
            p = -0.0060336708714301490533 + p * w;
            p = 0.24015818242558961693 + p * w;
            p = 1.6536545626831027356 + p * w;
        } else if (w < 16.000000) {
            w = sqrt(w) - 3.250000;
            p = 2.2137376921775787049e-09;
            p = 9.0756561938885390979e-08 + p * w;
            p = -2.7517406297064545428e-07 + p * w;
            p = 1.8239629214389227755e-08 + p * w;
            p = 1.5027403968909827627e-06 + p * w;
            p = -4.013867526981545969e-06 + p * w;
            p = 2.9234449089955446044e-06 + p * w;
            p = 1.2475304481671778723e-05 + p * w;
            p = -4.7318229009055733981e-05 + p * w;
            p = 6.8284851459573175448e-05 + p * w;
            p = 2.4031110387097893999e-05 + p * w;
            p = -0.0003550375203628474796 + p * w;
            p = 0.00095328937973738049703 + p * w;
            p = -0.0016882755560235047313 + p * w;
            p = 0.0024914420961078508066 + p * w;
            p = -0.0037512085075692412107 + p * w;
            p = 0.005370914553590063617 + p * w;
            p = 1.0052589676941592334 + p * w;
            p = 3.0838856104922207635 + p * w;
        } else {
            w = sqrt(w) - 5.000000;
            p = -2.7109920616438573243e-11;
            p = -2.5556418169965252055e-10 + p * w;
            p = 1.5076572693500548083e-09 + p * w;
            p = -3.7894654401267369937e-09 + p * w;
            p = 7.6157012080783393804e-09 + p * w;
            p = -1.4960026627149240478e-08 + p * w;
            p = 2.9147953450901080826e-08 + p * w;
            p = -6.7711997758452339498e-08 + p * w;
            p = 2.2900482228026654717e-07 + p * w;
            p = -9.9298272942317002539e-07 + p * w;
            p = 4.5260625972231537039e-06 + p * w;
            p = -1.9681778105531670567e-05 + p * w;
            p = 7.5995277030017761139e-05 + p * w;
            p = -0.00021503011930044477347 + p * w;
            p = -0.00013871931833623122026 + p * w;
            p = 1.0103004648645343977 + p * w;
            p = 4.8499064014085844221 + p * w;
        }
        return p * x;
    }

    double standard_deviation(std::vector<double>::iterator first, std::vector<double>::iterator last) {
        auto m = Catch::Benchmark::Detail::mean(first, last);
        double variance = std::accumulate(first, last, 0., [m](double a, double b) {
            double diff = b - m;
            return a + diff * diff;
            }) / (last - first);
            return std::sqrt(variance);
    }

}

namespace Catch {
    namespace Benchmark {
        namespace Detail {

            double weighted_average_quantile(int k, int q, std::vector<double>::iterator first, std::vector<double>::iterator last) {
                auto count = last - first;
                double idx = (count - 1) * k / static_cast<double>(q);
                int j = static_cast<int>(idx);
                double g = idx - j;
                std::nth_element(first, first + j, last);
                auto xj = first[j];
                if (g == 0) return xj;

                auto xj1 = *std::min_element(first + (j + 1), last);
                return xj + g * (xj1 - xj);
            }

            double erfc_inv(double x) {
                return erf_inv(1.0 - x);
            }

            double normal_quantile(double p) {
                static const double ROOT_TWO = std::sqrt(2.0);

                double result = 0.0;
                assert(p >= 0 && p <= 1);
                if (p < 0 || p > 1) {
                    return result;
                }

                result = -erfc_inv(2.0 * p);
                // result *= normal distribution standard deviation (1.0) * sqrt(2)
                result *= /*sd * */ ROOT_TWO;
                // result += normal disttribution mean (0)
                return result;
            }

            double outlier_variance(Estimate<double> mean, Estimate<double> stddev, int n) {
                double sb = stddev.point;
                double mn = mean.point / n;
                double mg_min = mn / 2.;
                double sg = (std::min)(mg_min / 4., sb / std::sqrt(n));
                double sg2 = sg * sg;
                double sb2 = sb * sb;

                auto c_max = [n, mn, sb2, sg2](double x) -> double {
                    double k = mn - x;
                    double d = k * k;
                    double nd = n * d;
                    double k0 = -n * nd;
                    double k1 = sb2 - n * sg2 + nd;
                    double det = k1 * k1 - 4 * sg2 * k0;
                    return (int)(-2. * k0 / (k1 + std::sqrt(det)));
                };

                auto var_out = [n, sb2, sg2](double c) {
                    double nc = n - c;
                    return (nc / n) * (sb2 - nc * sg2);
                };

                return (std::min)(var_out(1), var_out((std::min)(c_max(0.), c_max(mg_min)))) / sb2;
            }

            bootstrap_analysis analyse_samples(double confidence_level, int n_resamples,
      std::vector<double>::iterator first, std::vector<double>::iterator last) {
                CATCH_INTERNAL_START_WARNINGS_SUPPRESSION
                CATCH_INTERNAL_SUPPRESS_GLOBALS_WARNINGS
                static std::random_device entropy;
                CATCH_INTERNAL_STOP_WARNINGS_SUPPRESSION

                auto n = static_cast<int>(last - first); // seriously, one can't use integral types
      without hell in C++

                auto mean = &Detail::mean<std::vector<double>::iterator>;
                auto stddev = &standard_deviation;

#if defined(CATCH_CONFIG_USE_ASYNC)
                auto Estimate = [=](double(*f)(std::vector<double>::iterator,
      std::vector<double>::iterator)) {
                    auto seed = entropy();
                    return std::async(std::launch::async, [=] {
                        std::mt19937 rng(seed);
                        auto resampled = resample(rng, n_resamples, first, last, f);
                        return bootstrap(confidence_level, first, last, resampled, f);
                    });
                };

                auto mean_future = Estimate(mean);
                auto stddev_future = Estimate(stddev);

                auto mean_estimate = mean_future.get();
                auto stddev_estimate = stddev_future.get();
#else
                auto Estimate = [=](double(*f)(std::vector<double>::iterator,
      std::vector<double>::iterator)) {
                    auto seed = entropy();
                    std::mt19937 rng(seed);
                    auto resampled = resample(rng, n_resamples, first, last, f);
                    return bootstrap(confidence_level, first, last, resampled, f);
                };

                auto mean_estimate = Estimate(mean);
                auto stddev_estimate = Estimate(stddev);
#endif // CATCH_USE_ASYNC

                double outlier_variance = Detail::outlier_variance(mean_estimate, stddev_estimate, n);

                return { mean_estimate, stddev_estimate, outlier_variance };
            }
        } // namespace Detail
    } // namespace Benchmark
} // namespace Catch

#endif // CATCH_CONFIG_ENABLE_BENCHMARKING
// end catch_stats.cpp
// start catch_approx.cpp

#include <cmath>
#include <limits>

namespace {

// Performs equivalent check of std::fabs(lhs - rhs) <= margin
// But without the subtraction to allow for INFINITY in comparison
bool marginComparison(double lhs, double rhs, double margin) {
    return (lhs + margin >= rhs) && (rhs + margin >= lhs);
}

}

namespace Catch {
namespace Detail {

    Approx::Approx ( double value )
    :   m_epsilon( std::numeric_limits<float>::epsilon()*100 ),
        m_margin( 0.0 ),
        m_scale( 0.0 ),
        m_value( value )
    {}

    Approx Approx::custom() {
        return Approx( 0 );
    }

    Approx Approx::operator-() const {
        auto temp(*this);
        temp.m_value = -temp.m_value;
        return temp;
    }

    std::string Approx::toString() const {
        ReusableStringStream rss;
        rss « "Approx( " « ::Catch::Detail::stringify( m_value ) « " )";
        return rss.str();
    }

    bool Approx::equalityComparisonImpl(const double other) const {
        // First try with fixed margin, then compute margin based on epsilon, scale and Approx's value
        // Thanks to Richard Harris for his help refining the scaled margin value
        return marginComparison(m_value, other, m_margin)
            || marginComparison(m_value, other, m_epsilon * (m_scale + std::fabs(std::isinf(m_value)?
0 : m_value)));
    }

    void Approx::setMargin(double newMargin) {
        CATCH_ENFORCE(newMargin >= 0,
            "Invalid Approx::margin: " « newMargin « '.'
            « " Approx::Margin has to be non-negative.");
        m_margin = newMargin;
    }

    void Approx::setEpsilon(double newEpsilon) {
        CATCH_ENFORCE(newEpsilon >= 0 && newEpsilon <= 1.0,
            "Invalid Approx::epsilon: " « newEpsilon « '.'
            « " Approx::epsilon has to be in [0, 1]");
        m_epsilon = newEpsilon;
    }

} // end namespace Detail

namespace literals {
    Detail::Approx operator "" _a(long double val) {
        return Detail::Approx(val);
    }
    Detail::Approx operator "" _a(unsigned long long val) {
        return Detail::Approx(val);
    }
} // end namespace literals

std::string StringMaker<Catch::Detail::Approx>::convert(Catch::Detail::Approx const& value) {
    return value.toString();
}

} // end namespace Catch
// end catch_approx.cpp
// start catch_assertionhandler.cpp

// start catch_debugger.h

namespace Catch {
    bool isDebuggerActive();
}

#ifdef CATCH_PLATFORM_MAC

    #if defined(__i386__) || defined(__x86_64__)
        #define CATCH_TRAP() __asm__("int $3\n" : : ) /* NOLINT */
    #elif defined(__aarch64__)
        #define CATCH_TRAP()  __asm__(".inst 0xd43e0000")
    #endif

#elif defined(CATCH_PLATFORM_IPHONE)

    // use inline assembler
    #if defined(__i386__) || defined(__x86_64__)
        #define CATCH_TRAP()  __asm__("int $3")
    #elif defined(__aarch64__)
        #define CATCH_TRAP()  __asm__(".inst 0xd4200000")
    #elif defined(__arm__) && !defined(__thumb__)
        #define CATCH_TRAP()  __asm__(".inst 0xe7f001f0")
    #elif defined(__arm__) &&  defined(__thumb__)
        #define CATCH_TRAP()  __asm__(".inst 0xde01")
    #endif

#elif defined(CATCH_PLATFORM_LINUX)
    // If we can use inline assembler, do it because this allows us to break
    // directly at the location of the failing check instead of breaking inside
    // raise() called from it, i.e. one stack frame below.
    #if defined(__GNUC__) && (defined(__i386) || defined(__x86_64))
        #define CATCH_TRAP() asm volatile ("int $3") /* NOLINT */
    #else // Fall back to the generic way.
        #include <signal.h>

        #define CATCH_TRAP() raise(SIGTRAP)
    #endif
#elif defined(_MSC_VER)
    #define CATCH_TRAP() __debugbreak()
#elif defined(__MINGW32__)
    extern "C" __declspec(dllimport) void __stdcall DebugBreak();
    #define CATCH_TRAP() DebugBreak()
#endif

#ifndef CATCH_BREAK_INTO_DEBUGGER
    #ifdef CATCH_TRAP
        #define CATCH_BREAK_INTO_DEBUGGER() []{ if( Catch::isDebuggerActive() ) { CATCH_TRAP(); } }()
    #else
        #define CATCH_BREAK_INTO_DEBUGGER() []{}()
    #endif
#endif

// end catch_debugger.h
// start catch_run_context.h

// start catch_fatal_condition.h

#include <cassert>

namespace Catch {

    // Wrapper for platform-specific fatal error (signals/SEH) handlers
    //
    // Tries to be cooperative with other handlers, and not step over
    // other handlers. This means that unknown structured exceptions
    // are passed on, previous signal handlers are called, and so on.
    //
    // Can only be instantiated once, and assumes that once a signal
    // is caught, the binary will end up terminating. Thus, there
    class FatalConditionHandler {
        bool m_started = false;

        // Install/disengage implementation for specific platform.
        // Should be if-defed to work on current platform, can assume
        // engage-disengage 1:1 pairing.
        void engage_platform();
        void disengage_platform();
    public:
        // Should also have platform-specific implementations as needed
        FatalConditionHandler();
        ~FatalConditionHandler();

        void engage() {
            assert(!m_started && "Handler cannot be installed twice.");
            m_started = true;
            engage_platform();
        }

        void disengage() {
            assert(m_started && "Handler cannot be uninstalled without being installed first");
            m_started = false;
            disengage_platform();
        }
    };

    class FatalConditionHandlerGuard {
        FatalConditionHandler* m_handler;
    public:
        FatalConditionHandlerGuard(FatalConditionHandler* handler):
            m_handler(handler) {
            m_handler->engage();
        }
        ~FatalConditionHandlerGuard() {
            m_handler->disengage();
        }
    };

} // end namespace Catch

// end catch_fatal_condition.h
#include <string>

namespace Catch {

    struct IMutableContext;


    class RunContext : public IResultCapture, public IRunner {

    public:
        RunContext( RunContext const& ) = delete;
        RunContext& operator =( RunContext const& ) = delete;

        explicit RunContext( IConfigPtr const& _config, IStreamingReporterPtr&& reporter );

        ~RunContext() override;

        void testGroupStarting( std::string const& testSpec, std::size_t groupIndex, std::size_t
groupsCount );
        void testGroupEnded( std::string const& testSpec, Totals const& totals, std::size_t
groupIndex, std::size_t groupsCount );

        Totals runTest(TestCase const& testCase);

        IConfigPtr config() const;
        IStreamingReporter& reporter() const;

    public: // IResultCapture

        // Assertion handlers
        void handleExpr
                (   AssertionInfo const& info,
                    ITransientExpression const& expr,
                    AssertionReaction& reaction ) override;
        void handleMessage
                (   AssertionInfo const& info,
                    ResultWas::OfType resultType,
                    StringRef const& message,
                    AssertionReaction& reaction ) override;
        void handleUnexpectedExceptionNotThrown
                (   AssertionInfo const& info,
                    AssertionReaction& reaction ) override;
        void handleUnexpectedInflightException
                (   AssertionInfo const& info,
                    std::string const& message,
                    AssertionReaction& reaction ) override;
        void handleIncomplete
                (   AssertionInfo const& info ) override;
        void handleNonExpr
                (   AssertionInfo const &info,
                    ResultWas::OfType resultType,
                    AssertionReaction &reaction ) override;

        bool sectionStarted( SectionInfo const& sectionInfo, Counts& assertions ) override;

        void sectionEnded( SectionEndInfo const& endInfo ) override;
        void sectionEndedEarly( SectionEndInfo const& endInfo ) override;

        auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
IGeneratorTracker& override;

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
        void benchmarkPreparing( std::string const& name ) override;
        void benchmarkStarting( BenchmarkInfo const& info ) override;
        void benchmarkEnded( BenchmarkStats<> const& stats ) override;
        void benchmarkFailed( std::string const& error ) override;
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

        void pushScopedMessage( MessageInfo const& message ) override;
        void popScopedMessage( MessageInfo const& message ) override;

        void emplaceUnscopedMessage( MessageBuilder const& builder ) override;

        std::string getCurrentTestName() const override;

        const AssertionResult* getLastResult() const override;

        void exceptionEarlyReported() override;

        void handleFatalErrorCondition( StringRef message ) override;

        bool lastAssertionPassed() override;

        void assertionPassed() override;

    public:
        // !TBD We need to do this another way!
        bool aborting() const final;

    private:

        void runCurrentTest( std::string& redirectedCout, std::string& redirectedCerr );
        void invokeActiveTestCase();

        void resetAssertionInfo();
        bool testForMissingAssertions( Counts& assertions );

        void assertionEnded( AssertionResult const& result );
        void reportExpr
                (   AssertionInfo const &info,
                    ResultWas::OfType resultType,
                    ITransientExpression const *expr,
                    bool negated );

        void populateReaction( AssertionReaction& reaction );

    private:

        void handleUnfinishedSections();

        TestRunInfo m_runInfo;
        IMutableContext& m_context;
        TestCase const* m_activeTestCase = nullptr;
        ITracker* m_testCaseTracker = nullptr;
        Option<AssertionResult> m_lastResult;

        IConfigPtr m_config;
        Totals m_totals;
        IStreamingReporterPtr m_reporter;
        std::vector<MessageInfo> m_messages;
        std::vector<ScopedMessage> m_messageScopes; /* Keeps owners of so-called unscoped messages. */
        AssertionInfo m_lastAssertionInfo;
        std::vector<SectionEndInfo> m_unfinishedSections;
        std::vector<ITracker*> m_activeSections;
        TrackerContext m_trackerContext;
        FatalConditionHandler m_fatalConditionhandler;
        bool m_lastAssertionPassed = false;
        bool m_shouldReportUnexpected = true;
        bool m_includeSuccessfulResults;
    };

    void seedRng(IConfig const& config);
    unsigned int rngSeed();
} // end namespace Catch

// end catch_run_context.h
namespace Catch {

    namespace {
        auto operator «( std::ostream& os, ITransientExpression const& expr ) -> std::ostream& {
            expr.streamReconstructedExpression( os );
            return os;
        }
    }

    LazyExpression::LazyExpression( bool isNegated )
    :   m_isNegated( isNegated )
    {}

    LazyExpression::LazyExpression( LazyExpression const& other ) : m_isNegated( other.m_isNegated )
{}

    LazyExpression::operator bool() const {
        return m_transientExpression != nullptr;
    }

    auto operator « ( std::ostream& os, LazyExpression const& lazyExpr ) -> std::ostream& {
        if( lazyExpr.m_isNegated )
            os « "!";

        if( lazyExpr ) {
            if( lazyExpr.m_isNegated && lazyExpr.m_transientExpression->isBinaryExpression() )
                os « "(" « *lazyExpr.m_transientExpression « ")";
            else
                os « *lazyExpr.m_transientExpression;
        }
        else {
            os « "{** error - unchecked empty expression requested **}";
        }
        return os;
    }

    AssertionHandler::AssertionHandler
        (   StringRef const& macroName,
            SourceLineInfo const& lineInfo,
            StringRef capturedExpression,
            ResultDisposition::Flags resultDisposition )
    :   m_assertionInfo{ macroName, lineInfo, capturedExpression, resultDisposition },
        m_resultCapture( getResultCapture() )
    {}

    void AssertionHandler::handleExpr( ITransientExpression const& expr ) {
        m_resultCapture.handleExpr( m_assertionInfo, expr, m_reaction );
    }
    void AssertionHandler::handleMessage(ResultWas::OfType resultType, StringRef const& message) {
        m_resultCapture.handleMessage( m_assertionInfo, resultType, message, m_reaction );
    }

    auto AssertionHandler::allowThrows() const -> bool {
        return getCurrentContext().getConfig()->allowThrows();
    }

    void AssertionHandler::complete() {
        setCompleted();
        if( m_reaction.shouldDebugBreak ) {

            // If you find your debugger stopping you here then go one level up on the
            // call-stack for the code that caused it (typically a failed assertion)

            // (To go back to the test and change execution, jump over the throw, next)
            CATCH_BREAK_INTO_DEBUGGER();
        }
        if (m_reaction.shouldThrow) {
#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
            throw Catch::TestFailureException();
#else
            CATCH_ERROR( "Test failure requires aborting test!" );
#endif
        }
    }
    void AssertionHandler::setCompleted() {
        m_completed = true;
    }

    void AssertionHandler::handleUnexpectedInflightException() {
        m_resultCapture.handleUnexpectedInflightException( m_assertionInfo,
Catch::translateActiveException(), m_reaction );
    }

    void AssertionHandler::handleExceptionThrownAsExpected() {
        m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
    }
    void AssertionHandler::handleExceptionNotThrownAsExpected() {
        m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
    }

    void AssertionHandler::handleUnexpectedExceptionNotThrown() {
        m_resultCapture.handleUnexpectedExceptionNotThrown( m_assertionInfo, m_reaction );
    }

    void AssertionHandler::handleThrowingCallSkipped() {
        m_resultCapture.handleNonExpr(m_assertionInfo, ResultWas::Ok, m_reaction);
    }

    // This is the overload that takes a string and infers the Equals matcher from it
    // The more general overload, that takes any string matcher, is in catch_capture_matchers.cpp
    void handleExceptionMatchExpr( AssertionHandler& handler, std::string const& str, StringRef const&
matcherString  ) {
        handleExceptionMatchExpr( handler, Matchers::Equals( str ), matcherString );
    }

} // namespace Catch
// end catch_assertionhandler.cpp
// start catch_assertionresult.cpp

namespace Catch {
    AssertionResultData::AssertionResultData(ResultWas::OfType _resultType, LazyExpression const &
_lazyExpression):
        lazyExpression(_lazyExpression),
        resultType(_resultType) {}

    std::string AssertionResultData::reconstructExpression() const {

        if( reconstructedExpression.empty() ) {
            if( lazyExpression ) {
                ReusableStringStream rss;
                rss « lazyExpression;
                reconstructedExpression = rss.str();
            }
        }
        return reconstructedExpression;
    }

    AssertionResult::AssertionResult( AssertionInfo const& info, AssertionResultData const& data )
    :   m_info( info ),
        m_resultData( data )
    {}

    // Result was a success
    bool AssertionResult::succeeded() const {
        return Catch::isOk( m_resultData.resultType );
    }

    // Result was a success, or failure is suppressed
    bool AssertionResult::isOk() const {
        return Catch::isOk( m_resultData.resultType ) || shouldSuppressFailure(
m_info.resultDisposition );
    }

    ResultWas::OfType AssertionResult::getResultType() const {
        return m_resultData.resultType;
    }

    bool AssertionResult::hasExpression() const {
        return !m_info.capturedExpression.empty();
    }

    bool AssertionResult::hasMessage() const {
        return !m_resultData.message.empty();
    }

    std::string AssertionResult::getExpression() const {
        // Possibly overallocating by 3 characters should be basically free
        std::string expr; expr.reserve(m_info.capturedExpression.size() + 3);
        if (isFalseTest(m_info.resultDisposition)) {
            expr += "!(";
        }
        expr += m_info.capturedExpression;
        if (isFalseTest(m_info.resultDisposition)) {
            expr += ')';
        }
        return expr;
    }

    std::string AssertionResult::getExpressionInMacro() const {
        std::string expr;
        if( m_info.macroName.empty() )
            expr = static_cast<std::string>(m_info.capturedExpression);
        else {
            expr.reserve( m_info.macroName.size() + m_info.capturedExpression.size() + 4 );
            expr += m_info.macroName;
            expr += "( ";
            expr += m_info.capturedExpression;
            expr += " )";
        }
        return expr;
    }

    bool AssertionResult::hasExpandedExpression() const {
        return hasExpression() && getExpandedExpression() != getExpression();
    }

    std::string AssertionResult::getExpandedExpression() const {
        std::string expr = m_resultData.reconstructExpression();
        return expr.empty()
                ? getExpression()
                : expr;
    }

    std::string AssertionResult::getMessage() const {
        return m_resultData.message;
    }
    SourceLineInfo AssertionResult::getSourceInfo() const {
        return m_info.lineInfo;
    }

    StringRef AssertionResult::getTestMacroName() const {
        return m_info.macroName;
    }

} // end namespace Catch
// end catch_assertionresult.cpp
// start catch_capture_matchers.cpp

namespace Catch {

    using StringMatcher = Matchers::Impl::MatcherBase<std::string>;

    // This is the general overload that takes a any string matcher
    // There is another overload, in catch_assertionhandler.h/.cpp, that only takes a string and
infers
    // the Equals matcher (so the header does not mention matchers)
    void handleExceptionMatchExpr( AssertionHandler& handler, StringMatcher const& matcher, StringRef
const& matcherString  ) {
        std::string exceptionMessage = Catch::translateActiveException();
        MatchExpr<std::string, StringMatcher const&> expr( exceptionMessage, matcher, matcherString );
        handler.handleExpr( expr );
    }

} // namespace Catch
// end catch_capture_matchers.cpp
// start catch_commandline.cpp

// start catch_commandline.h

// start catch_clara.h

// Use Catch's value for console width (store Clara's off to the side, if present)
#ifdef CLARA_CONFIG_CONSOLE_WIDTH
#define CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
#undef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
#endif
#define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CONFIG_CONSOLE_WIDTH-1

#ifdef __clang__
#pragma clang diagnostic push
#pragma clang diagnostic ignored "-Wweak-vtables"
#pragma clang diagnostic ignored "-Wexit-time-destructors"
#pragma clang diagnostic ignored "-Wshadow"
#endif

// start clara.hpp
// Copyright 2017 Two Blue Cubes Ltd. All rights reserved.
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//
// See https://github.com/philsquared/Clara for more details

// Clara v1.1.5


#ifndef CATCH_CLARA_CONFIG_CONSOLE_WIDTH
#define CATCH_CLARA_CONFIG_CONSOLE_WIDTH 80
#endif

#ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
#define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_CLARA_CONFIG_CONSOLE_WIDTH
#endif

#ifndef CLARA_CONFIG_OPTIONAL_TYPE
#ifdef __has_include
#if __has_include(<optional>) && __cplusplus >= 201703L
#include <optional>
```

```cpp
#define CLARA_CONFIG_OPTIONAL_TYPE std::optional
#endif
#endif
#endif

// ----------- #included from clara_textflow.hpp -----------

// TextFlowCpp
//
// A single-header library for wrapping and laying out basic text, by Phil Nash
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//
// This project is hosted at https://github.com/philsquared/textflowcpp


#include <cassert>
#include <ostream>
#include <sstream>
#include <vector>

#ifndef CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH
#define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH 80
#endif

namespace Catch {
namespace clara {
namespace TextFlow {

inline auto isWhitespace(char c) -> bool {
    static std::string chars = " \t\n\r";
    return chars.find(c) != std::string::npos;
}
inline auto isBreakableBefore(char c) -> bool {
    static std::string chars = "[({<|";
    return chars.find(c) != std::string::npos;
}
inline auto isBreakableAfter(char c) -> bool {
    static std::string chars = "])}>.,:;*+-=&/\\";
    return chars.find(c) != std::string::npos;
}

class Columns;

class Column {
    std::vector<std::string> m_strings;
    size_t m_width = CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH;
    size_t m_indent = 0;
    size_t m_initialIndent = std::string::npos;

public:
    class iterator {
        friend Column;

        Column const& m_column;
        size_t m_stringIndex = 0;
        size_t m_pos = 0;

        size_t m_len = 0;
        size_t m_end = 0;
        bool m_suffix = false;

        iterator(Column const& column, size_t stringIndex)
            : m_column(column),
            m_stringIndex(stringIndex) {}

        auto line() const -> std::string const& { return m_column.m_strings[m_stringIndex]; }

        auto isBoundary(size_t at) const -> bool {
            assert(at > 0);
            assert(at <= line().size());

            return at == line().size() ||
                (isWhitespace(line()[at]) && !isWhitespace(line()[at - 1])) ||
                isBreakableBefore(line()[at]) ||
                isBreakableAfter(line()[at - 1]);
        }

        void calcLength() {
            assert(m_stringIndex < m_column.m_strings.size());

            m_suffix = false;
            auto width = m_column.m_width - indent();
            m_end = m_pos;
            if (line()[m_pos] == '\n') {
                ++m_end;
            }
            while (m_end < line().size() && line()[m_end] != '\n')
                ++m_end;

            if (m_end < m_pos + width) {
                m_len = m_end - m_pos;
            } else {
                size_t len = width;
                while (len > 0 && !isBoundary(m_pos + len))
                    --len;
                while (len > 0 && isWhitespace(line()[m_pos + len - 1]))
                    --len;

                if (len > 0) {
                    m_len = len;
                } else {
                    m_suffix = true;
                    m_len = width - 1;
                }
            }
        }

        auto indent() const -> size_t {
            auto initial = m_pos == 0 && m_stringIndex == 0 ? m_column.m_initialIndent :
std::string::npos;
            return initial == std::string::npos ? m_column.m_indent : initial;
        }

        auto addIndentAndSuffix(std::string const &plain) const -> std::string {
            return std::string(indent(), ' ') + (m_suffix ? plain + "-" : plain);
        }

    public:
        using difference_type = std::ptrdiff_t;
        using value_type = std::string;
        using pointer = value_type * ;
        using reference = value_type & ;
        using iterator_category = std::forward_iterator_tag;

        explicit iterator(Column const& column) : m_column(column) {
            assert(m_column.m_width > m_column.m_indent);
            assert(m_column.m_initialIndent == std::string::npos || m_column.m_width >
m_column.m_initialIndent);
            calcLength();
            if (m_len == 0)
                m_stringIndex++; // Empty string
        }

        auto operator *() const -> std::string {
            assert(m_stringIndex < m_column.m_strings.size());
            assert(m_pos <= m_end);
            return addIndentAndSuffix(line().substr(m_pos, m_len));
        }

        auto operator ++() -> iterator& {
            m_pos += m_len;
            if (m_pos < line().size() && line()[m_pos] == '\n')
                m_pos += 1;
            else
                while (m_pos < line().size() && isWhitespace(line()[m_pos]))
                    ++m_pos;

            if (m_pos == line().size()) {
                m_pos = 0;
                ++m_stringIndex;
            }
            if (m_stringIndex < m_column.m_strings.size())
                calcLength();
            return *this;
        }
        auto operator ++(int) -> iterator {
            iterator prev(*this);
            operator++();
            return prev;
        }

        auto operator ==(iterator const& other) const -> bool {
            return
                m_pos == other.m_pos &&
                m_stringIndex == other.m_stringIndex &&
                &m_column == &other.m_column;
        }
        auto operator !=(iterator const& other) const -> bool {
            return !operator==(other);
        }
    };
    using const_iterator = iterator;

    explicit Column(std::string const& text) { m_strings.push_back(text); }

    auto width(size_t newWidth) -> Column& {
        assert(newWidth > 0);
        m_width = newWidth;
        return *this;
    }
    auto indent(size_t newIndent) -> Column& {
        m_indent = newIndent;
        return *this;
    }
    auto initialIndent(size_t newIndent) -> Column& {
        m_initialIndent = newIndent;
        return *this;
    }

    auto width() const -> size_t { return m_width; }
    auto begin() const -> iterator { return iterator(*this); }
    auto end() const -> iterator { return { *this, m_strings.size() }; }

    inline friend std::ostream& operator « (std::ostream& os, Column const& col) {
        bool first = true;
        for (auto line : col) {
            if (first)
                first = false;
            else
                os « "\n";
            os « line;
        }
        return os;
    }

    auto operator + (Column const& other)->Columns;

    auto toString() const -> std::string {
        std::ostringstream oss;
        oss « *this;
        return oss.str();
    }
};

class Spacer : public Column {

public:
    explicit Spacer(size_t spaceWidth) : Column("") {
        width(spaceWidth);
    }
};

class Columns {
    std::vector<Column> m_columns;

public:

    class iterator {
        friend Columns;
        struct EndTag {};

        std::vector<Column> const& m_columns;
        std::vector<Column::iterator> m_iterators;
        size_t m_activeIterators;

        iterator(Columns const& columns, EndTag)
            : m_columns(columns.m_columns),
            m_activeIterators(0) {
            m_iterators.reserve(m_columns.size());

            for (auto const& col : m_columns)
                m_iterators.push_back(col.end());
        }

    public:
        using difference_type = std::ptrdiff_t;
        using value_type = std::string;
        using pointer = value_type * ;
        using reference = value_type & ;
        using iterator_category = std::forward_iterator_tag;

        explicit iterator(Columns const& columns)
            : m_columns(columns.m_columns),
            m_activeIterators(m_columns.size()) {
            m_iterators.reserve(m_columns.size());

            for (auto const& col : m_columns)
                m_iterators.push_back(col.begin());
        }

        auto operator ==(iterator const& other) const -> bool {
            return m_iterators == other.m_iterators;
        }
        auto operator !=(iterator const& other) const -> bool {
            return m_iterators != other.m_iterators;
        }
        auto operator *() const -> std::string {
            std::string row, padding;

            for (size_t i = 0; i < m_columns.size(); ++i) {
                auto width = m_columns[i].width();
                if (m_iterators[i] != m_columns[i].end()) {
                    std::string col = *m_iterators[i];
                    row += padding + col;
                    if (col.size() < width)
                        padding = std::string(width - col.size(), ' ');
                    else
                        padding = "";
                } else {
                    padding += std::string(width, ' ');
                }
            }
            return row;
        }
        auto operator ++() -> iterator& {
            for (size_t i = 0; i < m_columns.size(); ++i) {
                if (m_iterators[i] != m_columns[i].end())
                    ++m_iterators[i];
            }
            return *this;
        }
        auto operator ++(int) -> iterator {
            iterator prev(*this);
            operator++();
            return prev;
        }
    };
    using const_iterator = iterator;

    auto begin() const -> iterator { return iterator(*this); }
    auto end() const -> iterator { return { *this, iterator::EndTag() }; }

    auto operator += (Column const& col) -> Columns& {
        m_columns.push_back(col);
        return *this;
    }
    auto operator + (Column const& col) -> Columns {
        Columns combined = *this;
        combined += col;
        return combined;
    }

    inline friend std::ostream& operator « (std::ostream& os, Columns const& cols) {

        bool first = true;
        for (auto line : cols) {
            if (first)
                first = false;
            else
                os « "\n";
            os « line;
        }
        return os;
    }

    auto toString() const -> std::string {
        std::ostringstream oss;
        oss « *this;
        return oss.str();
    }
};

inline auto Column::operator + (Column const& other) -> Columns {
    Columns cols;
    cols += *this;
    cols += other;
    return cols;
}
}

}
}

// ----------- end of #include from clara_textflow.hpp -----------
// ........... back in clara.hpp

```

```
#include <cctype>
#include <string>
#include <memory>
#include <set>
#include <algorithm>

#if !defined(CATCH_PLATFORM_WINDOWS) && ( defined(WIN32) || defined(__WIN32__) || defined(_WIN32) ||
defined(_MSC_VER) )
#define CATCH_PLATFORM_WINDOWS
#endif

namespace Catch { namespace clara {
namespace detail {

    // Traits for extracting arg and return type of lambdas (for single argument lambdas)
    template<typename L>
    struct UnaryLambdaTraits : UnaryLambdaTraits<decltype( &L::operator() )> {};

    template<typename ClassT, typename ReturnT, typename... Args>
    struct UnaryLambdaTraits<ReturnT( ClassT::* )( Args... ) const> {
        static const bool isValid = false;
    };

    template<typename ClassT, typename ReturnT, typename ArgT>
    struct UnaryLambdaTraits<ReturnT( ClassT::* )( ArgT ) const> {
        static const bool isValid = true;
        using ArgType = typename std::remove_const<typename std::remove_reference<ArgT>::type>::type;
        using ReturnType = ReturnT;
    };

    class TokenStream;

    // Transport for raw args (copied from main args, or supplied via init list for testing)
    class Args {
        friend TokenStream;
        std::string m_exeName;
        std::vector<std::string> m_args;

    public:
        Args( int argc, char const* const* argv )
            : m_exeName(argv[0]),
              m_args(argv + 1, argv + argc) {}

        Args( std::initializer_list<std::string> args )
        :   m_exeName( *args.begin() ),
            m_args( args.begin()+1, args.end() )
        {}

        auto exeName() const -> std::string {
            return m_exeName;
        }
    };

    // Wraps a token coming from a token stream. These may not directly correspond to strings as a
single string
    // may encode an option + its argument if the : or = form is used
    enum class TokenType {
        Option, Argument
    };
    struct Token {
        TokenType type;
        std::string token;
    };

    inline auto isOptPrefix( char c ) -> bool {
        return c == '-'
#ifdef CATCH_PLATFORM_WINDOWS
            || c == '/'
#endif
        ;
    }

    // Abstracts iterators into args as a stream of tokens, with option arguments uniformly handled
    class TokenStream {
        using Iterator = std::vector<std::string>::const_iterator;
        Iterator it;
        Iterator itEnd;
        std::vector<Token> m_tokenBuffer;

        void loadBuffer() {
            m_tokenBuffer.resize( 0 );

            // Skip any empty strings
            while( it != itEnd && it->empty() )
                ++it;

            if( it != itEnd ) {
                auto const &next = *it;
                if( isOptPrefix( next[0] ) ) {
                    auto delimiterPos = next.find_first_of( " :=" );
                    if( delimiterPos != std::string::npos ) {
                        m_tokenBuffer.push_back( { TokenType::Option, next.substr( 0, delimiterPos ) }
);
                        m_tokenBuffer.push_back( { TokenType::Argument, next.substr( delimiterPos + 1
) } );
                    } else {
                        if( next[1] != '-' && next.size() > 2 ) {
                            std::string opt = "- ";
                            for( size_t i = 1; i < next.size(); ++i ) {
                                opt[1] = next[i];
                                m_tokenBuffer.push_back( { TokenType::Option, opt } );
                            }
                        } else {
                            m_tokenBuffer.push_back( { TokenType::Option, next } );
                        }
                    }
                } else {
                    m_tokenBuffer.push_back( { TokenType::Argument, next } );
                }
            }
        }

    public:
        explicit TokenStream( Args const &args ) : TokenStream( args.m_args.begin(), args.m_args.end()
) {}

        TokenStream( Iterator it, Iterator itEnd ) : it( it ), itEnd( itEnd ) {
            loadBuffer();
        }

        explicit operator bool() const {
            return !m_tokenBuffer.empty() || it != itEnd;
        }

        auto count() const -> size_t { return m_tokenBuffer.size() + (itEnd - it); }

        auto operator*() const -> Token {
            assert( !m_tokenBuffer.empty() );
            return m_tokenBuffer.front();
        }

        auto operator->() const -> Token const * {
            assert( !m_tokenBuffer.empty() );
            return &m_tokenBuffer.front();
        }

        auto operator++() -> TokenStream & {
            if( m_tokenBuffer.size() >= 2 ) {
                m_tokenBuffer.erase( m_tokenBuffer.begin() );
            } else {
                if( it != itEnd )
                    ++it;
                loadBuffer();
            }
            return *this;
        }
    };

    class ResultBase {
    public:
        enum Type {
            Ok, LogicError, RuntimeError
        };

    protected:
        ResultBase( Type type ) : m_type( type ) {}
        virtual ~ResultBase() = default;

        virtual void enforceOk() const = 0;

        Type m_type;
    };

    template<typename T>
    class ResultValueBase : public ResultBase {
    public:
        auto value() const -> T const & {
            enforceOk();
            return m_value;
        }

    protected:
        ResultValueBase( Type type ) : ResultBase( type ) {}

        ResultValueBase( ResultValueBase const &other ) : ResultBase( other ) {
            if( m_type == ResultBase::Ok )
                new( &m_value ) T( other.m_value );
        }

        ResultValueBase( Type, T const &value ) : ResultBase( Ok ) {
            new( &m_value ) T( value );
        }

        auto operator=( ResultValueBase const &other ) -> ResultValueBase & {
            if( m_type == ResultBase::Ok )
                m_value.~T();
            ResultBase::operator=(other);
            if( m_type == ResultBase::Ok )
                new( &m_value ) T( other.m_value );
            return *this;
        }

        ~ResultValueBase() override {
            if( m_type == Ok )
                m_value.~T();
        }

        union {
            T m_value;
        };
    };

    template<>
    class ResultValueBase<void> : public ResultBase {
    protected:
        using ResultBase::ResultBase;
    };

    template<typename T = void>
    class BasicResult : public ResultValueBase<T> {
    public:
        template<typename U>
        explicit BasicResult( BasicResult<U> const &other )
        :   ResultValueBase<T>( other.type() ),
            m_errorMessage( other.errorMessage() )
        {
            assert( type() != ResultBase::Ok );
        }

        template<typename U>
        static auto ok( U const &value ) -> BasicResult { return { ResultBase::Ok, value }; }
        static auto ok() -> BasicResult { return { ResultBase::Ok }; }
        static auto logicError( std::string const &message ) -> BasicResult { return {
      ResultBase::LogicError, message }; }
        static auto runtimeError( std::string const &message ) -> BasicResult { return {
      ResultBase::RuntimeError, message }; }

        explicit operator bool() const { return m_type == ResultBase::Ok; }
        auto type() const -> ResultBase::Type { return m_type; }
        auto errorMessage() const -> std::string { return m_errorMessage; }

    protected:
        void enforceOk() const override {

            // Errors shouldn't reach this point, but if they do
            // the actual error message will be in m_errorMessage
            assert( m_type != ResultBase::LogicError );
            assert( m_type != ResultBase::RuntimeError );
            if( m_type != ResultBase::Ok )
                std::abort();
        }

        std::string m_errorMessage; // Only populated if resultType is an error

        BasicResult( ResultBase::Type type, std::string const &message )
        :   ResultValueBase<T>(type),
            m_errorMessage(message)
        {
            assert( m_type != ResultBase::Ok );
        }

        using ResultValueBase<T>::ResultValueBase;
        using ResultBase::m_type;
    };

    enum class ParseResultType {
        Matched, NoMatch, ShortCircuitAll, ShortCircuitSame
    };

    class ParseState {
    public:

        ParseState( ParseResultType type, TokenStream const &remainingTokens )
        : m_type(type),
          m_remainingTokens( remainingTokens )
        {}

        auto type() const -> ParseResultType { return m_type; }
        auto remainingTokens() const -> TokenStream { return m_remainingTokens; }

    private:
        ParseResultType m_type;
        TokenStream m_remainingTokens;
    };

    using Result = BasicResult<void>;
    using ParserResult = BasicResult<ParseResultType>;
    using InternalParseResult = BasicResult<ParseState>;

    struct HelpColumns {
        std::string left;
        std::string right;
    };

    template<typename T>
    inline auto convertInto( std::string const &source, T& target ) -> ParserResult {
        std::stringstream ss;
        ss « source;
        ss » target;
        if( ss.fail() )
            return ParserResult::runtimeError( "Unable to convert '" + source + "' to destination type" );
        else
            return ParserResult::ok( ParseResultType::Matched );
    }
    inline auto convertInto( std::string const &source, std::string& target ) -> ParserResult {
        target = source;
        return ParserResult::ok( ParseResultType::Matched );
    }
    inline auto convertInto( std::string const &source, bool &target ) -> ParserResult {
        std::string srcLC = source;
        std::transform( srcLC.begin(), srcLC.end(), srcLC.begin(), []( unsigned char c ) { return static_cast<char>( std::tolower(c) ); } );
        if (srcLC == "y" || srcLC == "1" || srcLC == "true" || srcLC == "yes" || srcLC == "on")
            target = true;
        else if (srcLC == "n" || srcLC == "0" || srcLC == "false" || srcLC == "no" || srcLC == "off")
            target = false;
        else
            return ParserResult::runtimeError( "Expected a boolean value but did not recognise: '" + source + "'" );
        return ParserResult::ok( ParseResultType::Matched );
    }
#ifdef CLARA_CONFIG_OPTIONAL_TYPE
    template<typename T>
    inline auto convertInto( std::string const &source, CLARA_CONFIG_OPTIONAL_TYPE<T>& target ) -> ParserResult {
        T temp;
        auto result = convertInto( source, temp );
        if( result )
            target = std::move(temp);
        return result;
    }
#endif // CLARA_CONFIG_OPTIONAL_TYPE

    struct NonCopyable {
        NonCopyable() = default;
        NonCopyable( NonCopyable const & ) = delete;
        NonCopyable( NonCopyable && ) = delete;
        NonCopyable &operator=( NonCopyable const & ) = delete;
        NonCopyable &operator=( NonCopyable && ) = delete;
    };

    struct BoundRef : NonCopyable {
        virtual ~BoundRef() = default;
        virtual auto isContainer() const -> bool { return false; }
        virtual auto isFlag() const -> bool { return false; }
    };
    struct BoundValueRefBase : BoundRef {
        virtual auto setValue( std::string const &arg ) -> ParserResult = 0;
    };
    struct BoundFlagRefBase : BoundRef {
        virtual auto setFlag( bool flag ) -> ParserResult = 0;
        virtual auto isFlag() const -> bool { return true; }
    };

    template<typename T>
    struct BoundValueRef : BoundValueRefBase {
        T &m_ref;

        explicit BoundValueRef( T &ref ) : m_ref( ref ) {}

        auto setValue( std::string const &arg ) -> ParserResult override {
            return convertInto( arg, m_ref );
        }
    };

    template<typename T>
    struct BoundValueRef<std::vector<T>» : BoundValueRefBase {
        std::vector<T> &m_ref;

        explicit BoundValueRef( std::vector<T> &ref ) : m_ref( ref ) {}

        auto isContainer() const -> bool override { return true; }

        auto setValue( std::string const &arg ) -> ParserResult override {
            T temp;
            auto result = convertInto( arg, temp );
            if( result )
                m_ref.push_back( temp );
            return result;
        }
    };

    struct BoundFlagRef : BoundFlagRefBase {
        bool &m_ref;

        explicit BoundFlagRef( bool &ref ) : m_ref( ref ) {}

        auto setFlag( bool flag ) -> ParserResult override {
            m_ref = flag;
            return ParserResult::ok( ParseResultType::Matched );
        }
    };

    template<typename ReturnType>
    struct LambdaInvoker {
        static_assert( std::is_same<ReturnType, ParserResult>::value, "Lambda must return void or
clara::ParserResult" );

        template<typename L, typename ArgType>
        static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
            return lambda( arg );
        }
    };

    template<>
    struct LambdaInvoker<void> {
        template<typename L, typename ArgType>
        static auto invoke( L const &lambda, ArgType const &arg ) -> ParserResult {
            lambda( arg );
            return ParserResult::ok( ParseResultType::Matched );
        }
    };

    template<typename ArgType, typename L>
    inline auto invokeLambda( L const &lambda, std::string const &arg ) -> ParserResult {
        ArgType temp{};
        auto result = convertInto( arg, temp );
        return !result
           ? result
           : LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( lambda, temp );
    }

    template<typename L>
    struct BoundLambda : BoundValueRefBase {
        L m_lambda;

        static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument"
);
        explicit BoundLambda( L const &lambda ) : m_lambda( lambda ) {}

        auto setValue( std::string const &arg ) -> ParserResult override {
            return invokeLambda<typename UnaryLambdaTraits<L>::ArgType>( m_lambda, arg );
        }
    };

    template<typename L>
    struct BoundFlagLambda : BoundFlagRefBase {
        L m_lambda;

        static_assert( UnaryLambdaTraits<L>::isValid, "Supplied lambda must take exactly one argument"
);
        static_assert( std::is_same<typename UnaryLambdaTraits<L>::ArgType, bool>::value, "flags must
be boolean" );

```

```
        explicit BoundFlagLambda( L const &lambda ) : m_lambda( lambda ) {}

        auto setFlag( bool flag ) -> ParserResult override {
            return LambdaInvoker<typename UnaryLambdaTraits<L>::ReturnType>::invoke( m_lambda, flag );
        }
    };

    enum class Optionality { Optional, Required };

    struct Parser;

    class ParserBase {
    public:
        virtual ~ParserBase() = default;
        virtual auto validate() const -> Result { return Result::ok(); }
        virtual auto parse( std::string const& exeName, TokenStream const &tokens) const ->
InternalParseResult  = 0;
        virtual auto cardinality() const -> size_t { return 1; }

        auto parse( Args const &args ) const -> InternalParseResult {
            return parse( args.exeName(), TokenStream( args ) );
        }
    };

    template<typename DerivedT>
    class ComposableParserImpl : public ParserBase {
    public:
        template<typename T>
        auto operator|( T const &other ) const -> Parser;

        template<typename T>
        auto operator+( T const &other ) const -> Parser;
    };

    // Common code and state for Args and Opts
    template<typename DerivedT>
    class ParserRefImpl : public ComposableParserImpl<DerivedT> {
    protected:
        Optionality m_optionality = Optionality::Optional;
        std::shared_ptr<BoundRef> m_ref;
        std::string m_hint;
        std::string m_description;

        explicit ParserRefImpl( std::shared_ptr<BoundRef> const &ref ) : m_ref( ref ) {}

    public:
        template<typename T>
        ParserRefImpl( T &ref, std::string const &hint )
        :   m_ref( std::make_shared<BoundValueRef<T>»( ref ) ),
            m_hint( hint )
        {}

        template<typename LambdaT>
        ParserRefImpl( LambdaT const &ref, std::string const &hint )
        :   m_ref( std::make_shared<BoundLambda<LambdaT>»( ref ) ),
            m_hint(hint)
        {}

        auto operator()( std::string const &description ) -> DerivedT & {
            m_description = description;
            return static_cast<DerivedT &>( *this );
        }

        auto optional() -> DerivedT & {
            m_optionality = Optionality::Optional;
            return static_cast<DerivedT &>( *this );
        };

        auto required() -> DerivedT & {
            m_optionality = Optionality::Required;
            return static_cast<DerivedT &>( *this );
        };

        auto isOptional() const -> bool {
            return m_optionality == Optionality::Optional;
        }

        auto cardinality() const -> size_t override {
            if( m_ref->isContainer() )
                return 0;
            else
                return 1;
        }

        auto hint() const -> std::string { return m_hint; }
    };

    class ExeName : public ComposableParserImpl<ExeName> {
        std::shared_ptr<std::string> m_name;
        std::shared_ptr<BoundValueRefBase> m_ref;

        template<typename LambdaT>
        static auto makeRef(LambdaT const &lambda) -> std::shared_ptr<BoundValueRefBase> {
            return std::make_shared<BoundLambda<LambdaT»( lambda) ;
        }

    public:
        ExeName() : m_name( std::make_shared<std::string>( "<executable>" ) ) {}

        explicit ExeName( std::string &ref ) : ExeName() {
            m_ref = std::make_shared<BoundValueRef<std::string»( ref );
        }

        template<typename LambdaT>
        explicit ExeName( LambdaT const& lambda ) : ExeName() {
            m_ref = std::make_shared<BoundLambda<LambdaT»( lambda );
        }

        // The exe name is not parsed out of the normal tokens, but is handled specially
        auto parse( std::string const&, TokenStream const &tokens ) const -> InternalParseResult
override {
            return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
        }

        auto name() const -> std::string { return *m_name; }
        auto set( std::string const& newName ) -> ParserResult {

            auto lastSlash = newName.find_last_of( "\\/" );
            auto filename = ( lastSlash == std::string::npos )
                    ? newName
                    : newName.substr( lastSlash+1 );

            *m_name = filename;
            if( m_ref )
                return m_ref->setValue( filename );
            else
                return ParserResult::ok( ParseResultType::Matched );
        }
    };

    class Arg : public ParserRefImpl<Arg> {
    public:
        using ParserRefImpl::ParserRefImpl;

        auto parse( std::string const &, TokenStream const &tokens ) const -> InternalParseResult
override {
            auto validationResult = validate();
            if( !validationResult )
                return InternalParseResult( validationResult );

            auto remainingTokens = tokens;
            auto const &token = *remainingTokens;
            if( token.type != TokenType::Argument )
                return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens
) );

            assert( !m_ref->isFlag() );
            auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );

            auto result = valueRef->setValue( remainingTokens->token );
            if( !result )
                return InternalParseResult( result );
            else
                return InternalParseResult::ok( ParseState( ParseResultType::Matched,
++remainingTokens ) );
        }
    };

    inline auto normaliseOpt( std::string const &optName ) -> std::string {
#ifdef CATCH_PLATFORM_WINDOWS
        if( optName[0] == '/' )
            return "-" + optName.substr( 1 );
        else
#endif
            return optName;
    }

    class Opt : public ParserRefImpl<Opt> {
    protected:
        std::vector<std::string> m_optNames;

    public:
        template<typename LambdaT>
        explicit Opt( LambdaT const &ref ) : ParserRefImpl( std::make_shared<BoundFlagLambda<LambdaT»(
ref ) ) {}

        explicit Opt( bool &ref ) : ParserRefImpl( std::make_shared<BoundFlagRef>( ref ) ) {}

        template<typename LambdaT>
        Opt( LambdaT const &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}

        template<typename T>
        Opt( T &ref, std::string const &hint ) : ParserRefImpl( ref, hint ) {}

        auto operator[]( std::string const &optName ) -> Opt & {
            m_optNames.push_back( optName );
            return *this;
        }

        auto getHelpColumns() const -> std::vector<HelpColumns> {
            std::ostringstream oss;
            bool first = true;
            for( auto const &opt : m_optNames ) {
                if (first)
                    first = false;
                else
                    oss « ", ";
                oss « opt;
            }
            if( !m_hint.empty() )
                oss « " <" « m_hint « ">";
            return { { oss.str(), m_description } };
        }

        auto isMatch( std::string const &optToken ) const -> bool {
            auto normalisedToken = normaliseOpt( optToken );
            for( auto const &name : m_optNames ) {
                if( normaliseOpt( name ) == normalisedToken )
                    return true;
            }
            return false;
        }

        using ParserBase::parse;

        auto parse( std::string const&, TokenStream const &tokens ) const -> InternalParseResult
override {
            auto validationResult = validate();
            if( !validationResult )
                return InternalParseResult( validationResult );

            auto remainingTokens = tokens;
            if( remainingTokens && remainingTokens->type == TokenType::Option ) {
                auto const &token = *remainingTokens;
                if( isMatch(token.token ) ) {
                    if( m_ref->isFlag() ) {
                        auto flagRef = static_cast<detail::BoundFlagRefBase*>( m_ref.get() );
                        auto result = flagRef->setFlag( true );
                        if( !result )
                            return InternalParseResult( result );
                        if( result.value() == ParseResultType::ShortCircuitAll )
                            return InternalParseResult::ok( ParseState( result.value(),
remainingTokens ) );
                    } else {
                        auto valueRef = static_cast<detail::BoundValueRefBase*>( m_ref.get() );
                        ++remainingTokens;
                        if( !remainingTokens )
                            return InternalParseResult::runtimeError( "Expected argument following " +
token.token );
                        auto const &argToken = *remainingTokens;
                        if( argToken.type != TokenType::Argument )
                            return InternalParseResult::runtimeError( "Expected argument following " +
token.token );
                        auto result = valueRef->setValue( argToken.token );
                        if( !result )
                            return InternalParseResult( result );
                        if( result.value() == ParseResultType::ShortCircuitAll )
                            return InternalParseResult::ok( ParseState( result.value(),
remainingTokens ) );
                    }
                    return InternalParseResult::ok( ParseState( ParseResultType::Matched,
++remainingTokens ) );
                }
            }
            return InternalParseResult::ok( ParseState( ParseResultType::NoMatch, remainingTokens ) );
        }

        auto validate() const -> Result override {
            if( m_optNames.empty() )
                return Result::logicError( "No options supplied to Opt" );
            for( auto const &name : m_optNames ) {
                if( name.empty() )
                    return Result::logicError( "Option name cannot be empty" );
#ifdef CATCH_PLATFORM_WINDOWS
                if( name[0] != '-' && name[0] != '/' )
                    return Result::logicError( "Option name must begin with '-' or '/'" );
#else
                if( name[0] != '-' )
                    return Result::logicError( "Option name must begin with '-'" );
#endif
            }
            return ParserRefImpl::validate();
        }
    };

    struct Help : Opt {
        Help( bool &showHelpFlag )
        :   Opt([&]( bool flag ) {
                showHelpFlag = flag;
                return ParserResult::ok( ParseResultType::ShortCircuitAll );
            })
        {
            static_cast<Opt &>( *this )
                    ("display usage information")
                    ["-?"]["-h"]["--help"]
                    .optional();
        }
    };

    struct Parser : ParserBase {

        mutable ExeName m_exeName;
        std::vector<Opt> m_options;
        std::vector<Arg> m_args;

        auto operator|=( ExeName const &exeName ) -> Parser & {
            m_exeName = exeName;
            return *this;
        }

        auto operator|=( Arg const &arg ) -> Parser & {
            m_args.push_back(arg);
            return *this;
        }

        auto operator|=( Opt const &opt ) -> Parser & {
            m_options.push_back(opt);
            return *this;
        }

        auto operator|=( Parser const &other ) -> Parser & {
            m_options.insert(m_options.end(), other.m_options.begin(), other.m_options.end());
            m_args.insert(m_args.end(), other.m_args.begin(), other.m_args.end());
            return *this;
        }

        template<typename T>
        auto operator|( T const &other ) const -> Parser {
            return Parser( *this ) |= other;
        }

        // Forward deprecated interface with '+' instead of '|'
        template<typename T>
        auto operator+=( T const &other ) -> Parser & { return operator|=( other ); }
        template<typename T>
        auto operator+( T const &other ) const -> Parser { return operator|( other ); }

        auto getHelpColumns() const -> std::vector<HelpColumns> {
            std::vector<HelpColumns> cols;
            for (auto const &o : m_options) {
                auto childCols = o.getHelpColumns();
                cols.insert( cols.end(), childCols.begin(), childCols.end() );
            }
            return cols;
        }

        void writeToStream( std::ostream &os ) const {
            if (!m_exeName.name().empty()) {
                os « "usage:\n" « "  " « m_exeName.name() « " ";
                bool required = true, first = true;
                for( auto const &arg : m_args ) {
                    if (first)
                        first = false;
                    else
                        os « " ";
                    if( arg.isOptional() && required ) {
                        os « "[";
                        required = false;
                    }
                    os « "<" « arg.hint() « ">";
                    if( arg.cardinality() == 0 )
                        os « " ... ";
                }
                if( !required )
                    os « "]";
                if( !m_options.empty() )
                    os « " options";
                os « "\n\nwhere options are:" « std::endl;
            }

            auto rows = getHelpColumns();
            size_t consoleWidth = CATCH_CLARA_CONFIG_CONSOLE_WIDTH;
            size_t optWidth = 0;
            for( auto const &cols : rows )
                optWidth = (std::max)(optWidth, cols.left.size() + 2);

            optWidth = (std::min)(optWidth, consoleWidth/2);

            for( auto const &cols : rows ) {
                auto row =
                        TextFlow::Column( cols.left ).width( optWidth ).indent( 2 ) +
                        TextFlow::Spacer(4) +
                        TextFlow::Column( cols.right ).width( consoleWidth - 7 - optWidth );
                os « row « std::endl;
            }
        }

        friend auto operator«( std::ostream &os, Parser const &parser ) -> std::ostream& {
            parser.writeToStream( os );
            return os;
        }

        auto validate() const -> Result override {
            for( auto const &opt : m_options ) {
                auto result = opt.validate();
                if( !result )
                    return result;
            }
            for( auto const &arg : m_args ) {
                auto result = arg.validate();
                if( !result )
                    return result;
            }
            return Result::ok();
        }

        using ParserBase::parse;

        auto parse( std::string const& exeName, TokenStream const &tokens ) const -> InternalParseResult override {

            struct ParserInfo {
                ParserBase const* parser = nullptr;
                size_t count = 0;
            };
            const size_t totalParsers = m_options.size() + m_args.size();
            assert( totalParsers < 512 );
            // ParserInfo parseInfos[totalParsers]; // <-- this is what we really want to do
            ParserInfo parseInfos[512];

            {
                size_t i = 0;
                for (auto const &opt : m_options) parseInfos[i++].parser = &opt;
                for (auto const &arg : m_args) parseInfos[i++].parser = &arg;
            }

            m_exeName.set( exeName );

            auto result = InternalParseResult::ok( ParseState( ParseResultType::NoMatch, tokens ) );
            while( result.value().remainingTokens() ) {
                bool tokenParsed = false;

                for( size_t i = 0; i < totalParsers; ++i ) {
                    auto&  parseInfo = parseInfos[i];
                    if( parseInfo.parser->cardinality() == 0 || parseInfo.count < parseInfo.parser->cardinality() ) {
                        result = parseInfo.parser->parse(exeName, result.value().remainingTokens());
                        if (!result)
                            return result;
                        if (result.value().type() != ParseResultType::NoMatch) {
                            tokenParsed = true;
                            ++parseInfo.count;
                            break;
                        }
                    }
                }

                if( result.value().type() == ParseResultType::ShortCircuitAll )
                    return result;
                if( !tokenParsed )
                    return InternalParseResult::runtimeError( "Unrecognised token: " +
result.value().remainingTokens()->token );
            }
            // !TBD Check missing required options
            return result;
        }
    };

    template<typename DerivedT>
    template<typename T>
    auto ComposableParserImpl<DerivedT>::operator|( T const &other ) const -> Parser {
        return Parser() | static_cast<DerivedT const &>( *this ) | other;
    }
} // namespace detail

// A Combined parser
using detail::Parser;

// A parser for options
using detail::Opt;

// A parser for arguments
using detail::Arg;

// Wrapper for argc, argv from main()
using detail::Args;

// Specifies the name of the executable
using detail::ExeName;

// Convenience wrapper for option parser that specifies the help option
using detail::Help;

// enum of result types from a parse
using detail::ParseResultType;

// Result type for parser operation
using detail::ParserResult;

}} // namespace Catch::clara

// end clara.hpp
#ifdef __clang__
#pragma clang diagnostic pop
#endif

// Restore Clara's value for console width, if present
#ifdef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
#define CATCH_CLARA_TEXTFLOW_CONFIG_CONSOLE_WIDTH CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
#undef CATCH_TEMP_CLARA_CONFIG_CONSOLE_WIDTH
#endif

// end catch_clara.h
namespace Catch {

    clara::Parser makeCommandLineParser( ConfigData& config );

} // end namespace Catch

// end catch_commandline.h
#include <fstream>
#include <ctime>

namespace Catch {

    clara::Parser makeCommandLineParser( ConfigData& config ) {

        using namespace clara;

        auto const setWarning = [&]( std::string const& warning ) {
                auto warningSet = [&]() {
                    if( warning == "NoAssertions" )
                        return WarnAbout::NoAssertions;

                    if ( warning == "NoTests" )
                        return WarnAbout::NoTests;

                    return WarnAbout::Nothing;
                }();

                if (warningSet == WarnAbout::Nothing)
                    return ParserResult::runtimeError( "Unrecognised warning: '" + warning + "'" );
                config.warnings = static_cast<WarnAbout::What>( config.warnings | warningSet );
                return ParserResult::ok( ParseResultType::Matched );
            };
        auto const loadTestNamesFromFile = [&]( std::string const& filename ) {
                std::ifstream f( filename.c_str() );
                if( !f.is_open() )
                    return ParserResult::runtimeError( "Unable to load input file: '" + filename + "'"
);

                std::string line;
                while( std::getline( f, line ) ) {
                    line = trim(line);
                    if( !line.empty() && !startsWith( line, '#' ) ) {
                        if( !startsWith( line, '"' ) )
                            line = '"' + line + '"';
                        config.testsOrTags.push_back( line );
                        config.testsOrTags.emplace_back( "," );
                    }
                }
                //Remove comma in the end
                if(!config.testsOrTags.empty())
                    config.testsOrTags.erase( config.testsOrTags.end()-1 );

                return ParserResult::ok( ParseResultType::Matched );
            };
        auto const setTestOrder = [&]( std::string const& order ) {
                if( startsWith( "declared", order ) )
                    config.runOrder = RunTests::InDeclarationOrder;
                else if( startsWith( "lexical", order ) )
                    config.runOrder = RunTests::InLexicographicalOrder;
                else if( startsWith( "random", order ) )
                    config.runOrder = RunTests::InRandomOrder;
                else
                    return clara::ParserResult::runtimeError( "Unrecognised ordering: '" + order + "'"
);
                return ParserResult::ok( ParseResultType::Matched );
            };
        auto const setRngSeed = [&]( std::string const& seed ) {
                if( seed != "time" )
                    return clara::detail::convertInto( seed, config.rngSeed );
                config.rngSeed = static_cast<unsigned int>( std::time(nullptr) );
                return ParserResult::ok( ParseResultType::Matched );
            };
        auto const setColourUsage = [&]( std::string const& useColour ) {
                    auto mode = toLower( useColour );

                    if( mode == "yes" )
                        config.useColour = UseColour::Yes;
                    else if( mode == "no" )
                        config.useColour = UseColour::No;
                    else if( mode == "auto" )
                        config.useColour = UseColour::Auto;
                    else
                        return ParserResult::runtimeError( "colour mode must be one of: auto, yes or
no. '" + useColour + "' not recognised" );
                return ParserResult::ok( ParseResultType::Matched );
            };
        auto const setWaitForKeypress = [&]( std::string const& keypress ) {
                auto keypressLc = toLower( keypress );
                if (keypressLc == "never")
                    config.waitForKeypress = WaitForKeypress::Never;
                else if( keypressLc == "start" )
                    config.waitForKeypress = WaitForKeypress::BeforeStart;
                else if( keypressLc == "exit" )
                    config.waitForKeypress = WaitForKeypress::BeforeExit;
                else if( keypressLc == "both" )
                    config.waitForKeypress = WaitForKeypress::BeforeStartAndExit;
                else
                    return ParserResult::runtimeError( "keypress argument must be one of: never,
start, exit or both. '" + keypress + "' not recognised" );
            return ParserResult::ok( ParseResultType::Matched );
            };
        auto const setVerbosity = [&]( std::string const& verbosity ) {
            auto lcVerbosity = toLower( verbosity );
            if( lcVerbosity == "quiet" )
                config.verbosity = Verbosity::Quiet;
            else if( lcVerbosity == "normal" )
                config.verbosity = Verbosity::Normal;
            else if( lcVerbosity == "high" )
                config.verbosity = Verbosity::High;
            else
                return ParserResult::runtimeError( "Unrecognised verbosity, '" + verbosity + "'" );
            return ParserResult::ok( ParseResultType::Matched );
        };
        auto const setReporter = [&]( std::string const& reporter ) {
            IReporterRegistry::FactoryMap const& factories =
getRegistryHub().getReporterRegistry().getFactories();
```

```

            auto lcReporter = toLower( reporter );
            auto result = factories.find( lcReporter );

            if( factories.end() != result )
                config.reporterName = lcReporter;
            else
                return ParserResult::runtimeError( "Unrecognized reporter, '" + reporter + "'. Check
      available with --list-reporters" );
            return ParserResult::ok( ParseResultType::Matched );
        };

        auto cli
            = ExeName( config.processName )
            | Help( config.showHelp )
            | Opt( config.listTests )
                ["-l"]["--list-tests"]
                ( "list all/matching test cases" )
            | Opt( config.listTags )
                ["-t"]["--list-tags"]
                ( "list all/matching tags" )
            | Opt( config.showSuccessfulTests )
                ["-s"]["--success"]
                ( "include successful tests in output" )
            | Opt( config.shouldDebugBreak )
                ["-b"]["--break"]
                ( "break into debugger on failure" )
            | Opt( config.noThrow )
                ["-e"]["--nothrow"]
                ( "skip exception tests" )
            | Opt( config.showInvisibles )
                ["-i"]["--invisibles"]
                ( "show invisibles (tabs, newlines)" )
            | Opt( config.outputFilename, "filename" )
                ["-o"]["--out"]
                ( "output filename" )
            | Opt( setReporter, "name" )
                ["-r"]["--reporter"]
                ( "reporter to use (defaults to console)" )
            | Opt( config.name, "name" )
                ["-n"]["--name"]
                ( "suite name" )
            | Opt( [&]( bool ){ config.abortAfter = 1; } )
                ["-a"]["--abort"]
                ( "abort at first failure" )
            | Opt( [&]( int x ){ config.abortAfter = x; }, "no. failures" )
                ["-x"]["--abortx"]
                ( "abort after x failures" )
            | Opt( setWarning, "warning name" )
                ["-w"]["--warn"]
                ( "enable warnings" )
            | Opt( [&]( bool flag ) { config.showDurations = flag ? ShowDurations::Always :
      ShowDurations::Never; }, "yes|no" )
                ["-d"]["--durations"]
                ( "show test durations" )
            | Opt( config.minDuration, "seconds" )
                ["-D"]["--min-duration"]
                ( "show test durations for tests taking at least the given number of seconds" )
            | Opt( loadTestNamesFromFile, "filename" )
                ["-f"]["--input-file"]
                ( "load test names to run from a file" )
            | Opt( config.filenamesAsTags )
                ["-#"]["--filenames-as-tags"]
                ( "adds a tag for the filename" )
            | Opt( config.sectionsToRun, "section name" )
                ["-c"]["--section"]
                ( "specify section to run" )
            | Opt( setVerbosity, "quiet|normal|high" )
                ["-v"]["--verbosity"]
                ( "set output verbosity" )
            | Opt( config.listTestNamesOnly )
                ["--list-test-names-only"]
                ( "list all/matching test cases names only" )
            | Opt( config.listReporters )
                ["--list-reporters"]
                ( "list all reporters" )
            | Opt( setTestOrder, "decl|lex|rand" )
                ["--order"]
                ( "test case order (defaults to decl)" )
            | Opt( setRngSeed, "'time'|number" )
                ["--rng-seed"]
                ( "set a specific seed for random numbers" )
            | Opt( setColourUsage, "yes|no" )
                ["--use-colour"]
                ( "should output be colourised" )
            | Opt( config.libIdentify )
                ["--libidentify"]
                ( "report name and version according to libidentify standard" )
            | Opt( setWaitForKeypress, "never|start|exit|both" )
                ["--wait-for-keypress"]
                ( "waits for a keypress before exiting" )
            | Opt( config.benchmarkSamples, "samples" )
                ["--benchmark-samples"]
                ( "number of samples to collect (default: 100)" )
            | Opt( config.benchmarkResamples, "resamples" )
                ["--benchmark-resamples"]
                ( "number of resamples for the bootstrap (default: 100000)" )
            | Opt( config.benchmarkConfidenceInterval, "confidence interval" )
                ["--benchmark-confidence-interval"]
                ( "confidence interval for the bootstrap (between 0 and 1, default: 0.95)" )
            | Opt( config.benchmarkNoAnalysis )
                ["--benchmark-no-analysis"]
                ( "perform only measurements; do not perform any analysis" )
            | Opt( config.benchmarkWarmupTime, "benchmarkWarmupTime" )
                ["--benchmark-warmup-time"]
                ( "amount of time in milliseconds spent on warming up each test (default: 100)" )
            | Arg( config.testsOrTags, "test name|pattern|tags" )
                ( "which test or tests to use" );

        return cli;
    }

} // end namespace Catch
// end catch_commandline.cpp
// start catch_common.cpp

#include <cstring>
#include <ostream>

namespace Catch {

    bool SourceLineInfo::operator == ( SourceLineInfo const& other ) const noexcept {
        return line == other.line && (file == other.file || std::strcmp(file, other.file) == 0);
    }
    bool SourceLineInfo::operator < ( SourceLineInfo const& other ) const noexcept {
        // We can assume that the same file will usually have the same pointer.
        // Thus, if the pointers are the same, there is no point in calling the strcmp
        return line < other.line || ( line == other.line && file != other.file && (std::strcmp(file,
other.file) < 0));
    }

    std::ostream& operator « ( std::ostream& os, SourceLineInfo const& info ) {
#ifndef __GNUG__
        os « info.file « '(' « info.line « ')';
#else
        os « info.file « ':' « info.line;
#endif
        return os;
    }

    std::string StreamEndStop::operator+() const {
        return std::string();
    }

    NonCopyable::NonCopyable() = default;
    NonCopyable::~NonCopyable() = default;

}
// end catch_common.cpp
// start catch_config.cpp

namespace Catch {

    Config::Config( ConfigData const& data )
    :   m_data( data ),
        m_stream( openStream() )
    {
        // We need to trim filter specs to avoid trouble with superfluous
        // whitespace (esp. important for bdd macros, as those are manually
        // aligned with whitespace).

        for (auto& elem : m_data.testsOrTags) {
            elem = trim(elem);
        }
        for (auto& elem : m_data.sectionsToRun) {
            elem = trim(elem);
        }

        TestSpecParser parser(ITagAliasRegistry::get());
        if (!m_data.testsOrTags.empty()) {
            m_hasTestFilters = true;
            for (auto const& testOrTags : m_data.testsOrTags) {
                parser.parse(testOrTags);
            }
        }
        m_testSpec = parser.testSpec();
    }

    std::string const& Config::getFilename() const {
        return m_data.outputFilename ;
    }

    bool Config::listTests() const          { return m_data.listTests; }
    bool Config::listTestNamesOnly() const  { return m_data.listTestNamesOnly; }
    bool Config::listTags() const           { return m_data.listTags; }
    bool Config::listReporters() const      { return m_data.listReporters; }

    std::string Config::getProcessName() const { return m_data.processName; }
    std::string const& Config::getReporterName() const { return m_data.reporterName; }

    std::vector<std::string> const& Config::getTestsOrTags() const { return m_data.testsOrTags; }
    std::vector<std::string> const& Config::getSectionsToRun() const { return m_data.sectionsToRun; }

    TestSpec const& Config::testSpec() const { return m_testSpec; }
    bool Config::hasTestFilters() const { return m_hasTestFilters; }

    bool Config::showHelp() const { return m_data.showHelp; }

    // IConfig interface
    bool Config::allowThrows() const                   { return !m_data.noThrow; }
    std::ostream& Config::stream() const               { return m_stream->stream(); }
    std::string Config::name() const                   { return m_data.name.empty() ? m_data.processName : m_data.name; }
    bool Config::includeSuccessfulResults() const      { return m_data.showSuccessfulTests; }
    bool Config::warnAboutMissingAssertions() const    { return !!(m_data.warnings & WarnAbout::NoAssertions); }
    bool Config::warnAboutNoTests() const              { return !!(m_data.warnings & WarnAbout::NoTests); }
    ShowDurations::OrNot Config::showDurations() const { return m_data.showDurations; }
    double Config::minDuration() const                 { return m_data.minDuration; }
    RunTests::InWhatOrder Config::runOrder() const     { return m_data.runOrder; }
    unsigned int Config::rngSeed() const               { return m_data.rngSeed; }
    UseColour::YesOrNo Config::useColour() const       { return m_data.useColour; }
    bool Config::shouldDebugBreak() const              { return m_data.shouldDebugBreak; }
    int Config::abortAfter() const                     { return m_data.abortAfter; }
    bool Config::showInvisibles() const                { return m_data.showInvisibles; }
    Verbosity Config::verbosity() const                { return m_data.verbosity; }

    bool Config::benchmarkNoAnalysis() const                      { return m_data.benchmarkNoAnalysis; }
    int Config::benchmarkSamples() const                          { return m_data.benchmarkSamples; }
    double Config::benchmarkConfidenceInterval() const            { return m_data.benchmarkConfidenceInterval; }
    unsigned int Config::benchmarkResamples() const               { return m_data.benchmarkResamples; }
    std::chrono::milliseconds Config::benchmarkWarmupTime() const { return std::chrono::milliseconds(m_data.benchmarkWarmupTime); }

    IStream const* Config::openStream() {
        return Catch::makeStream(m_data.outputFilename);
    }

} // end namespace Catch
// end catch_config.cpp
// start catch_console_colour.cpp

#if defined(__clang__)
#    pragma clang diagnostic push
#    pragma clang diagnostic ignored "-Wexit-time-destructors"
#endif

// start catch_errno_guard.h

namespace Catch {

    class ErrnoGuard {
    public:
        ErrnoGuard();
        ~ErrnoGuard();
    private:
        int m_oldErrno;
    };

}

// end catch_errno_guard.h
// start catch_windows_h_proxy.h


#if defined(CATCH_PLATFORM_WINDOWS)

```

```
#if !defined(NOMINMAX) && !defined(CATCH_CONFIG_NO_NOMINMAX)
#  define CATCH_DEFINED_NOMINMAX
#  define NOMINMAX
#endif
#if !defined(WIN32_LEAN_AND_MEAN) && !defined(CATCH_CONFIG_NO_WIN32_LEAN_AND_MEAN)
#  define CATCH_DEFINED_WIN32_LEAN_AND_MEAN
#  define WIN32_LEAN_AND_MEAN
#endif

#ifdef __AFXDLL
#include <AfxWin.h>
#else
#include <windows.h>
#endif

#ifdef CATCH_DEFINED_NOMINMAX
#  undef NOMINMAX
#endif
#ifdef CATCH_DEFINED_WIN32_LEAN_AND_MEAN
#  undef WIN32_LEAN_AND_MEAN
#endif

#endif // defined(CATCH_PLATFORM_WINDOWS)

// end catch_windows_h_proxy.h
#include <sstream>

namespace Catch {
    namespace {

        struct IColourImpl {
            virtual ~IColourImpl() = default;
            virtual void use( Colour::Code _colourCode ) = 0;
        };

        struct NoColourImpl : IColourImpl {
            void use( Colour::Code ) override {}

            static IColourImpl* instance() {
                static NoColourImpl s_instance;
                return &s_instance;
            }
        };

    } // anon namespace
} // namespace Catch

#if !defined( CATCH_CONFIG_COLOUR_NONE ) && !defined( CATCH_CONFIG_COLOUR_WINDOWS ) && !defined(
CATCH_CONFIG_COLOUR_ANSI )
#   ifdef CATCH_PLATFORM_WINDOWS
#       define CATCH_CONFIG_COLOUR_WINDOWS
#   else
#       define CATCH_CONFIG_COLOUR_ANSI
#   endif
#endif

#if defined ( CATCH_CONFIG_COLOUR_WINDOWS )

namespace Catch {
namespace {

    class Win32ColourImpl : public IColourImpl {
    public:
        Win32ColourImpl() : stdoutHandle( GetStdHandle(STD_OUTPUT_HANDLE) )
        {
            CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
            GetConsoleScreenBufferInfo( stdoutHandle, &csbiInfo );
            originalForegroundAttributes = csbiInfo.wAttributes & ~( BACKGROUND_GREEN | BACKGROUND_RED
| BACKGROUND_BLUE | BACKGROUND_INTENSITY );
            originalBackgroundAttributes = csbiInfo.wAttributes & ~( FOREGROUND_GREEN | FOREGROUND_RED
| FOREGROUND_BLUE | FOREGROUND_INTENSITY );
        }

        void use( Colour::Code _colourCode ) override {
            switch( _colourCode ) {
                case Colour::None:      return setTextAttribute( originalForegroundAttributes );
                case Colour::White:     return setTextAttribute( FOREGROUND_GREEN | FOREGROUND_RED |
FOREGROUND_BLUE );
                case Colour::Red:       return setTextAttribute( FOREGROUND_RED );
                case Colour::Green:     return setTextAttribute( FOREGROUND_GREEN );
                case Colour::Blue:      return setTextAttribute( FOREGROUND_BLUE );
                case Colour::Cyan:      return setTextAttribute( FOREGROUND_BLUE | FOREGROUND_GREEN );
                case Colour::Yellow:    return setTextAttribute( FOREGROUND_RED | FOREGROUND_GREEN );
                case Colour::Grey:      return setTextAttribute( 0 );

                case Colour::LightGrey:     return setTextAttribute( FOREGROUND_INTENSITY );
                case Colour::BrightRed:     return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED );
                case Colour::BrightGreen:   return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN );
                case Colour::BrightWhite:   return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE );
                case Colour::BrightYellow:  return setTextAttribute( FOREGROUND_INTENSITY |
FOREGROUND_RED | FOREGROUND_GREEN );

                case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );

                default:
                    CATCH_ERROR( "Unknown colour requested" );
            }
        }

    private:
        void setTextAttribute( WORD _textAttribute ) {
            SetConsoleTextAttribute( stdoutHandle, _textAttribute | originalBackgroundAttributes );
        }
        HANDLE stdoutHandle;
        WORD originalForegroundAttributes;
        WORD originalBackgroundAttributes;
    };

    IColourImpl* platformColourInstance() {
        static Win32ColourImpl s_instance;

        IConfigPtr config = getCurrentContext().getConfig();
        UseColour::YesOrNo colourMode = config
            ? config->useColour()
            : UseColour::Auto;
        if( colourMode == UseColour::Auto )
            colourMode = UseColour::Yes;
        return colourMode == UseColour::Yes
            ? &s_instance
            : NoColourImpl::instance();
    }

} // end anon namespace
} // end namespace Catch

#elif defined( CATCH_CONFIG_COLOUR_ANSI )

#include <unistd.h>

namespace Catch {
namespace {

    // use POSIX/ ANSI console terminal codes
    // Thanks to Adam Strzelecki for original contribution
    // (http://github.com/nanoant)
    // https://github.com/philsquared/Catch/pull/131
    class PosixColourImpl : public IColourImpl {
    public:
        void use( Colour::Code _colourCode ) override {
            switch( _colourCode ) {
                case Colour::None:
                case Colour::White:     return setColour( "[0m" );
                case Colour::Red:       return setColour( "[0;31m" );
                case Colour::Green:     return setColour( "[0;32m" );
                case Colour::Blue:      return setColour( "[0;34m" );
                case Colour::Cyan:      return setColour( "[0;36m" );
                case Colour::Yellow:    return setColour( "[0;33m" );
                case Colour::Grey:      return setColour( "[1;30m" );

                case Colour::LightGrey:     return setColour( "[0;37m" );
                case Colour::BrightRed:     return setColour( "[1;31m" );
                case Colour::BrightGreen:   return setColour( "[1;32m" );
                case Colour::BrightWhite:   return setColour( "[1;37m" );
                case Colour::BrightYellow:  return setColour( "[1;33m" );

                case Colour::Bright: CATCH_INTERNAL_ERROR( "not a colour" );
                default: CATCH_INTERNAL_ERROR( "Unknown colour requested" );
            }
        }
        static IColourImpl* instance() {
            static PosixColourImpl s_instance;
            return &s_instance;
        }

    private:
        void setColour( const char* _escapeCode ) {
            getCurrentContext().getConfig()->stream()
                « '\033' « _escapeCode;
        }
    };

    bool useColourOnPlatform() {
        return
#if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)
            !isDebuggerActive() &&
#endif
#if !(defined(__DJGPP__) && defined(__STRICT_ANSI__))
            isatty(STDOUT_FILENO)
#else
            false
#endif
            ;
    }
    IColourImpl* platformColourInstance() {
        ErrnoGuard guard;
        IConfigPtr config = getCurrentContext().getConfig();
        UseColour::YesOrNo colourMode = config
            ? config->useColour()
            : UseColour::Auto;
        if( colourMode == UseColour::Auto )
            colourMode = useColourOnPlatform()
                ? UseColour::Yes
                : UseColour::No;
        return colourMode == UseColour::Yes
            ? PosixColourImpl::instance()
            : NoColourImpl::instance();
    }

} // end anon namespace
} // end namespace Catch

#else  // not Windows or ANSI ///////////////////////////////////////////////

namespace Catch {

    static IColourImpl* platformColourInstance() { return NoColourImpl::instance(); }

} // end namespace Catch

#endif // Windows/ ANSI/ None

namespace Catch {

    Colour::Colour( Code _colourCode ) { use( _colourCode ); }
    Colour::Colour( Colour&& other ) noexcept {
        m_moved = other.m_moved;
        other.m_moved = true;
    }
    Colour& Colour::operator=( Colour&& other ) noexcept {
        m_moved = other.m_moved;
        other.m_moved  = true;
        return *this;
    }

    Colour::~Colour(){ if( !m_moved ) use( None ); }

    void Colour::use( Code _colourCode ) {
        static IColourImpl* impl = platformColourInstance();
        // Strictly speaking, this cannot possibly happen.
        // However, under some conditions it does happen (see #1626),
        // and this change is small enough that we can let practicality
        // triumph over purity in this case.
        if (impl != nullptr) {
            impl->use( _colourCode );
        }
    }

    std::ostream& operator « ( std::ostream& os, Colour const& ) {
        return os;
    }

} // end namespace Catch

#if defined(__clang__)
#    pragma clang diagnostic pop
#endif

// end catch_console_colour.cpp
// start catch_context.cpp

namespace Catch {

    class Context : public IMutableContext, NonCopyable {

    public: // IContext
        IResultCapture* getResultCapture() override {
            return m_resultCapture;
        }
        IRunner* getRunner() override {
            return m_runner;
        }

        IConfigPtr const& getConfig() const override {
            return m_config;
        }

        ~Context() override;

    public: // IMutableContext
        void setResultCapture( IResultCapture* resultCapture ) override {
            m_resultCapture = resultCapture;
        }
        void setRunner( IRunner* runner ) override {
            m_runner = runner;
        }
        void setConfig( IConfigPtr const& config ) override {
            m_config = config;
        }

        friend IMutableContext& getCurrentMutableContext();

    private:
        IConfigPtr m_config;
        IRunner* m_runner = nullptr;
        IResultCapture* m_resultCapture = nullptr;
    };

    IMutableContext *IMutableContext::currentContext = nullptr;

    void IMutableContext::createContext()
    {
        currentContext = new Context();
    }

    void cleanUpContext() {
        delete IMutableContext::currentContext;
        IMutableContext::currentContext = nullptr;
    }
    IContext::~IContext() = default;
    IMutableContext::~IMutableContext() = default;
    Context::~Context() = default;

    SimplePcg32& rng() {
        static SimplePcg32 s_rng;
        return s_rng;
    }

}
// end catch_context.cpp
// start catch_debug_console.cpp

// start catch_debug_console.h

#include <string>

namespace Catch {
    void writeToDebugConsole( std::string const& text );
}

// end catch_debug_console.h
#if defined(CATCH_CONFIG_ANDROID_LOGWRITE)
#include <android/log.h>

    namespace Catch {
        void writeToDebugConsole( std::string const& text ) {
            __android_log_write( ANDROID_LOG_DEBUG, "Catch", text.c_str() );
        }
    }

#elif defined(CATCH_PLATFORM_WINDOWS)

    namespace Catch {
        void writeToDebugConsole( std::string const& text ) {
            ::OutputDebugStringA( text.c_str() );
        }
    }

#else

    namespace Catch {
        void writeToDebugConsole( std::string const& text ) {
            // !TBD: Need a version for Mac/ XCode and other IDEs
            Catch::cout() « text;
        }
    }
```

```

#endif // Platform
// end catch_debug_console.cpp
// start catch_debugger.cpp

#if defined(CATCH_PLATFORM_MAC) || defined(CATCH_PLATFORM_IPHONE)

#  include <cassert>
#  include <sys/types.h>
#  include <unistd.h>
#  include <cstddef>
#  include <ostream>

#ifdef __apple_build_version__
    // These headers will only compile with AppleClang (XCode)
    // For other compilers (Clang, GCC, ... ) we need to exclude them
#  include <sys/sysctl.h>
#endif

    namespace Catch {
        #ifdef __apple_build_version__
        // The following function is taken directly from the following technical note:
        // https://developer.apple.com/library/archive/qa/qa1361/_index.html

        // Returns true if the current process is being debugged (either
        // running under the debugger or has a debugger attached post facto).
        bool isDebuggerActive(){
            int                 mib[4];
            struct kinfo_proc   info;
            std::size_t         size;

            // Initialize the flags so that, if sysctl fails for some bizarre
            // reason, we get a predictable result.

            info.kp_proc.p_flag = 0;

            // Initialize mib, which tells sysctl the info we want, in this case
            // we're looking for information about a specific process ID.

            mib[0] = CTL_KERN;
            mib[1] = KERN_PROC;
            mib[2] = KERN_PROC_PID;
            mib[3] = getpid();

            // Call sysctl.

            size = sizeof(info);
            if( sysctl(mib, sizeof(mib) / sizeof(*mib), &info, &size, nullptr, 0) != 0 ) {
                Catch::cerr() « "\n** Call to sysctl failed - unable to determine if debugger is
active **\n" « std::endl;
                return false;
            }

            // We're being debugged if the P_TRACED flag is set.

            return ( (info.kp_proc.p_flag & P_TRACED) != 0 );
        }
        #else
        bool isDebuggerActive() {
            // We need to find another way to determine this for non-appleclang compilers on macOS
            return false;
        }
        #endif
    } // namespace Catch

#elif defined(CATCH_PLATFORM_LINUX)
    #include <fstream>
    #include <string>

    namespace Catch{
        // The standard POSIX way of detecting a debugger is to attempt to
        // ptrace() the process, but this needs to be done from a child and not
        // this process itself to still allow attaching to this process later
        // if wanted, so is rather heavy. Under Linux we have the PID of the
        // "debugger" (which doesn't need to be gdb, of course, it could also
        // be strace, for example) in /proc/$PID/status, so just get it from
        // there instead.
        bool isDebuggerActive(){
            // Libstdc++ has a bug, where std::ifstream sets errno to 0
            // This way our users can properly assert over errno values
            ErrnoGuard guard;
            std::ifstream in("/proc/self/status");
            for( std::string line; std::getline(in, line); ) {
                static const int PREFIX_LEN = 11;
                if( line.compare(0, PREFIX_LEN, "TracerPid:\t") == 0 ) {
                    // We're traced if the PID is not 0 and no other PID starts
                    // with 0 digit, so it's enough to check for just a single
                    // character.
                    return line.length() > PREFIX_LEN && line[PREFIX_LEN] != '0';
                }
            }

            return false;
        }
    } // namespace Catch
#elif defined(_MSC_VER)
    extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
    namespace Catch {
        bool isDebuggerActive() {
            return IsDebuggerPresent() != 0;
        }
    }
#elif defined(__MINGW32__)
    extern "C" __declspec(dllimport) int __stdcall IsDebuggerPresent();
    namespace Catch {
        bool isDebuggerActive() {
            return IsDebuggerPresent() != 0;
        }
    }
#else
    namespace Catch {
       bool isDebuggerActive() { return false; }
    }
#endif // Platform
// end catch_debugger.cpp
// start catch_decomposer.cpp

namespace Catch {

    ITransientExpression::~ITransientExpression() = default;

    void formatReconstructedExpression( std::ostream &os, std::string const& lhs, StringRef op,
std::string const& rhs ) {
        if( lhs.size() + rhs.size() < 40 &&
                lhs.find('\n') == std::string::npos &&
                rhs.find('\n') == std::string::npos )
            os « lhs « " " « op « " " « rhs;
        else
            os « lhs « "\n" « op « "\n" « rhs;
    }
}
// end catch_decomposer.cpp
// start catch_enforce.cpp

#include <stdexcept>

namespace Catch {
#if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS) &&
!defined(CATCH_CONFIG_DISABLE_EXCEPTIONS_CUSTOM_HANDLER)
    [[noreturn]]
    void throw_exception(std::exception const& e) {
        Catch::cerr() « "Catch will terminate because it needed to throw an exception.\n"
                      « "The message was: " « e.what() « '\n';
        std::terminate();
    }
#endif

    [[noreturn]]
    void throw_logic_error(std::string const& msg) {
        throw_exception(std::logic_error(msg));
    }

    [[noreturn]]
    void throw_domain_error(std::string const& msg) {
        throw_exception(std::domain_error(msg));
    }

    [[noreturn]]
    void throw_runtime_error(std::string const& msg) {
        throw_exception(std::runtime_error(msg));
    }

} // namespace Catch;
// end catch_enforce.cpp
// start catch_enum_values_registry.cpp
// start catch_enum_values_registry.h

#include <vector>
#include <memory>

namespace Catch {

    namespace Detail {

        std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
std::vector<int> const& values );

        class EnumValuesRegistry : public IMutableEnumValuesRegistry {

            std::vector<std::unique_ptr<EnumInfo» m_enumInfos;

            EnumInfo const& registerEnum( StringRef enumName, StringRef allEnums, std::vector<int>
const& values) override;
        };

        std::vector<StringRef> parseEnums( StringRef enums );

    } // Detail

} // Catch

// end catch_enum_values_registry.h

#include <map>
#include <cassert>

namespace Catch {

    IMutableEnumValuesRegistry::~IMutableEnumValuesRegistry() {}

    namespace Detail {

        namespace {
            // Extracts the actual name part of an enum instance
            // In other words, it returns the Blue part of Bikeshed::Colour::Blue
            StringRef extractInstanceName(StringRef enumInstance) {
                // Find last occurrence of ":"
                size_t name_start = enumInstance.size();
                while (name_start > 0 && enumInstance[name_start - 1] != ':') {
                    --name_start;
                }
                return enumInstance.substr(name_start, enumInstance.size() - name_start);
            }
        }

        std::vector<StringRef> parseEnums( StringRef enums ) {
            auto enumValues = splitStringRef( enums, ',' );
            std::vector<StringRef> parsed;
            parsed.reserve( enumValues.size() );
            for( auto const& enumValue : enumValues ) {
                parsed.push_back(trim(extractInstanceName(enumValue)));
            }
            return parsed;
        }

        EnumInfo::~EnumInfo() {}

        StringRef EnumInfo::lookup( int value ) const {
            for( auto const& valueToName : m_values ) {
                if( valueToName.first == value )
                    return valueToName.second;
            }
            return "{** unexpected enum value **}"_sr;
        }

        std::unique_ptr<EnumInfo> makeEnumInfo( StringRef enumName, StringRef allValueNames,
std::vector<int> const& values ) {
            std::unique_ptr<EnumInfo> enumInfo( new EnumInfo );
            enumInfo->m_name = enumName;
            enumInfo->m_values.reserve( values.size() );

            const auto valueNames = Catch::Detail::parseEnums( allValueNames );
            assert( valueNames.size() == values.size() );
            std::size_t i = 0;
            for( auto value : values )
                enumInfo->m_values.emplace_back(value, valueNames[i++]);

            return enumInfo;
        }

        EnumInfo const& EnumValuesRegistry::registerEnum( StringRef enumName, StringRef allValueNames,
std::vector<int> const& values ) {
            m_enumInfos.push_back(makeEnumInfo(enumName, allValueNames, values));
            return *m_enumInfos.back();
        }

    } // Detail
} // Catch

// end catch_enum_values_registry.cpp
// start catch_errno_guard.cpp
```

```

#include <cerrno>

namespace Catch {
        ErrnoGuard::ErrnoGuard():m_oldErrno(errno){}
        ErrnoGuard::~ErrnoGuard() { errno = m_oldErrno; }
}
// end catch_errno_guard.cpp
// start catch_exception_translator_registry.cpp

// start catch_exception_translator_registry.h

#include <vector>
#include <string>
#include <memory>

namespace Catch {

    class ExceptionTranslatorRegistry : public IExceptionTranslatorRegistry {
    public:
        ~ExceptionTranslatorRegistry();
        virtual void registerTranslator( const IExceptionTranslator* translator );
        std::string translateActiveException() const override;
        std::string tryTranslators() const;

    private:
        std::vector<std::unique_ptr<IExceptionTranslator const» m_translators;
    };
}

// end catch_exception_translator_registry.h
#ifdef __OBJC__
#import "Foundation/Foundation.h"
#endif

namespace Catch {

    ExceptionTranslatorRegistry::~ExceptionTranslatorRegistry() {
    }

    void ExceptionTranslatorRegistry::registerTranslator( const IExceptionTranslator* translator ) {
        m_translators.push_back( std::unique_ptr<const IExceptionTranslator>( translator ) );
    }

#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
    std::string ExceptionTranslatorRegistry::translateActiveException() const {
        try {
#ifdef __OBJC__
            // In Objective-C try objective-c exceptions first
            @try {
                return tryTranslators();
            }
            @catch (NSException *exception) {
                return Catch::Detail::stringify( [exception description] );
            }
#else
            // Compiling a mixed mode project with MSVC means that CLR
            // exceptions will be caught in (...) as well. However, these
            // do not fill-in std::current_exception and thus lead to crash
            // when attempting rethrow.
            // /EHa switch also causes structured exceptions to be caught
            // here, but they fill-in current_exception properly, so
            // at worst the output should be a little weird, instead of
            // causing a crash.
            if (std::current_exception() == nullptr) {
                return "Non C++ exception. Possibly a CLR exception.";
            }
            return tryTranslators();
#endif
        }
        catch( TestFailureException& ) {
            std::rethrow_exception(std::current_exception());
        }
        catch( std::exception& ex ) {
            return ex.what();
        }
        catch( std::string& msg ) {
            return msg;
        }
        catch( const char* msg ) {
            return msg;
        }
        catch(...) {
            return "Unknown exception";
        }
    }

    std::string ExceptionTranslatorRegistry::tryTranslators() const {
        if (m_translators.empty()) {
            std::rethrow_exception(std::current_exception());
        } else {
            return m_translators[0]->translate(m_translators.begin() + 1, m_translators.end());
        }
    }

#else // ^^ Exceptions are enabled // Exceptions are disabled vv
    std::string ExceptionTranslatorRegistry::translateActiveException() const {
        CATCH_INTERNAL_ERROR("Attempted to translate active exception under
CATCH_CONFIG_DISABLE_EXCEPTIONS!");
    }

    std::string ExceptionTranslatorRegistry::tryTranslators() const {
        CATCH_INTERNAL_ERROR("Attempted to use exception translators under
CATCH_CONFIG_DISABLE_EXCEPTIONS!");
    }
#endif

}
// end catch_exception_translator_registry.cpp
// start catch_fatal_condition.cpp

#include <algorithm>

#if !defined( CATCH_CONFIG_WINDOWS_SEH ) && !defined( CATCH_CONFIG_POSIX_SIGNALS )

namespace Catch {

    // If neither SEH nor signal handling is required, the handler impls
    // do not have to do anything, and can be empty.
    void FatalConditionHandler::engage_platform() {}
    void FatalConditionHandler::disengage_platform() {}
    FatalConditionHandler::FatalConditionHandler() = default;
    FatalConditionHandler::~FatalConditionHandler() = default;

} // end namespace Catch

#endif // !CATCH_CONFIG_WINDOWS_SEH && !CATCH_CONFIG_POSIX_SIGNALS

#if defined( CATCH_CONFIG_WINDOWS_SEH ) && defined( CATCH_CONFIG_POSIX_SIGNALS )
#error "Inconsistent configuration: Windows' SEH handling and POSIX signals cannot be enabled at the
same time"
#endif // CATCH_CONFIG_WINDOWS_SEH && CATCH_CONFIG_POSIX_SIGNALS

#if defined( CATCH_CONFIG_WINDOWS_SEH ) || defined( CATCH_CONFIG_POSIX_SIGNALS )

namespace {
    void reportFatal( char const * const message ) {
        Catch::getCurrentContext().getResultCapture()->handleFatalErrorCondition( message );
    }

    constexpr std::size_t minStackSizeForErrors = 32 * 1024;
} // end unnamed namespace

#endif // CATCH_CONFIG_WINDOWS_SEH || CATCH_CONFIG_POSIX_SIGNALS

#if defined( CATCH_CONFIG_WINDOWS_SEH )

namespace Catch {

    struct SignalDefs { DWORD id; const char* name; };

    // There is no 1-1 mapping between signals and windows exceptions.
    // Windows can easily distinguish between SO and SigSegV,
    // but SigInt, SigTerm, etc are handled differently.
    static SignalDefs signalDefs[] = {
        { static_cast<DWORD>(EXCEPTION_ILLEGAL_INSTRUCTION),  "SIGILL - Illegal instruction signal" },
        { static_cast<DWORD>(EXCEPTION_STACK_OVERFLOW), "SIGSEGV - Stack overflow" },
        { static_cast<DWORD>(EXCEPTION_ACCESS_VIOLATION), "SIGSEGV - Segmentation violation signal" },
        { static_cast<DWORD>(EXCEPTION_INT_DIVIDE_BY_ZERO), "Divide by zero error" },
    };

    static LONG CALLBACK handleVectoredException(PEXCEPTION_POINTERS ExceptionInfo) {
        for (auto const& def : signalDefs) {
            if (ExceptionInfo->ExceptionRecord->ExceptionCode == def.id) {
                reportFatal(def.name);
            }
        }
        // If its not an exception we care about, pass it along.
        // This stops us from eating debugger breaks etc.
        return EXCEPTION_CONTINUE_SEARCH;
    }

    // Since we do not support multiple instantiations, we put these
    // into global variables and rely on cleaning them up in outlined
    // constructors/destructors
    static PVOID exceptionHandlerHandle = nullptr;

    // For MSVC, we reserve part of the stack memory for handling
    // memory overflow structured exception.
    FatalConditionHandler::FatalConditionHandler() {
        ULONG guaranteeSize = static_cast<ULONG>(minStackSizeForErrors);
        if (!SetThreadStackGuarantee(&guaranteeSize)) {
            // We do not want to fully error out, because needing
            // the stack reserve should be rare enough anyway.
            Catch::cerr()
                « "Failed to reserve piece of stack."
                « " Stack overflows will not be reported successfully.";
        }
    }

    // We do not attempt to unset the stack guarantee, because
    // Windows does not support lowering the stack size guarantee.
    FatalConditionHandler::~FatalConditionHandler() = default;

    void FatalConditionHandler::engage_platform() {
        // Register as first handler in current chain
        exceptionHandlerHandle = AddVectoredExceptionHandler(1, handleVectoredException);
        if (!exceptionHandlerHandle) {
            CATCH_RUNTIME_ERROR("Could not register vectored exception handler");
        }
    }

    void FatalConditionHandler::disengage_platform() {
        if (!RemoveVectoredExceptionHandler(exceptionHandlerHandle)) {
            CATCH_RUNTIME_ERROR("Could not unregister vectored exception handler");
        }
        exceptionHandlerHandle = nullptr;
    }

} // end namespace Catch

#endif // CATCH_CONFIG_WINDOWS_SEH

#if defined( CATCH_CONFIG_POSIX_SIGNALS )

#include <signal.h>

namespace Catch {

    struct SignalDefs {
        int id;
        const char* name;
    };

    static SignalDefs signalDefs[] = {
        { SIGINT,  "SIGINT - Terminal interrupt signal" },
        { SIGILL,  "SIGILL - Illegal instruction signal" },
        { SIGFPE,  "SIGFPE - Floating point error signal" },
        { SIGSEGV, "SIGSEGV - Segmentation violation signal" },
        { SIGTERM, "SIGTERM - Termination request signal" },
        { SIGABRT, "SIGABRT - Abort (abnormal termination) signal" }
    };

// Older GCCs trigger -Wmissing-field-initializers for T foo = {}
// which is zero initialization, but not explicit. We want to avoid
// that.
#if defined(__GNUC__)
#    pragma GCC diagnostic push
#    pragma GCC diagnostic ignored "-Wmissing-field-initializers"
#endif

    static char* altStackMem = nullptr;
    static std::size_t altStackSize = 0;
    static stack_t oldSigStack{};
    static struct sigaction oldSigActions[sizeof(signalDefs) / sizeof(SignalDefs)]{};

    static void restorePreviousSignalHandlers() {
        // We set signal handlers back to the previous ones. Hopefully
        // nobody overwrote them in the meantime, and doesn't expect
        // their signal handlers to live past ours given that they
        // installed them after ours..
        for (std::size_t i = 0; i < sizeof(signalDefs) / sizeof(SignalDefs); ++i) {
            sigaction(signalDefs[i].id, &oldSigActions[i], nullptr);
        }
        // Return the old stack
        sigaltstack(&oldSigStack, nullptr);
    }

    static void handleSignal( int sig ) {
        char const * name = "<unknown signal>";
        for (auto const& def : signalDefs) {
            if (sig == def.id) {
                name = def.name;
                break;
            }
        }
        // We need to restore previous signal handlers and let them do
        // their thing, so that the users can have the debugger break
        // when a signal is raised, and so on.
        restorePreviousSignalHandlers();
        reportFatal( name );
        raise( sig );
    }

    FatalConditionHandler::FatalConditionHandler() {
        assert(!altStackMem && "Cannot initialize POSIX signal handler when one already exists");
        if (altStackSize == 0) {
            altStackSize = std::max(static_cast<size_t>(SIGSTKSZ), minStackSizeForErrors);
        }
        altStackMem = new char[altStackSize]();
    }

    FatalConditionHandler::~FatalConditionHandler() {
        delete[] altStackMem;
        // We signal that another instance can be constructed by zeroing
        // out the pointer.
        altStackMem = nullptr;
    }

    void FatalConditionHandler::engage_platform() {
        stack_t sigStack;
        sigStack.ss_sp = altStackMem;
        sigStack.ss_size = altStackSize;
        sigStack.ss_flags = 0;
        sigaltstack(&sigStack, &oldSigStack);
        struct sigaction sa = { };

        sa.sa_handler = handleSignal;
        sa.sa_flags = SA_ONSTACK;
        for (std::size_t i = 0; i < sizeof(signalDefs)/sizeof(SignalDefs); ++i) {
            sigaction(signalDefs[i].id, &sa, &oldSigActions[i]);
        }
    }

#if defined(__GNUC__)
#    pragma GCC diagnostic pop
#endif

    void FatalConditionHandler::disengage_platform() {
        restorePreviousSignalHandlers();
    }

} // end namespace Catch

#endif // CATCH_CONFIG_POSIX_SIGNALS
// end catch_fatal_condition.cpp
// start catch_generators.cpp

#include <limits>
#include <set>

namespace Catch {

IGeneratorTracker::~IGeneratorTracker() {}

const char* GeneratorException::what() const noexcept {
    return m_msg;
}

namespace Generators {

    GeneratorUntypedBase::~GeneratorUntypedBase() {}

    auto acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo ) ->
IGeneratorTracker& {
        return getResultCapture().acquireGeneratorTracker( generatorName, lineInfo );
    }

} // namespace Generators
} // namespace Catch
// end catch_generators.cpp
// start catch_interfaces_capture.cpp

namespace Catch {
    IResultCapture::~IResultCapture() = default;
}
// end catch_interfaces_capture.cpp
// start catch_interfaces_config.cpp
```

```

namespace Catch {
    IConfig::~IConfig() = default;
}
// end catch_interfaces_config.cpp
// start catch_interfaces_exception.cpp

namespace Catch {
    IExceptionTranslator::~IExceptionTranslator() = default;
    IExceptionTranslatorRegistry::~IExceptionTranslatorRegistry() = default;
}
// end catch_interfaces_exception.cpp
// start catch_interfaces_registry_hub.cpp

namespace Catch {
    IRegistryHub::~IRegistryHub() = default;
    IMutableRegistryHub::~IMutableRegistryHub() = default;
}
// end catch_interfaces_registry_hub.cpp
// start catch_interfaces_reporter.cpp

// start catch_reporter_listening.h

namespace Catch {

    class ListeningReporter : public IStreamingReporter {
        using Reporters = std::vector<IStreamingReporterPtr>;
        Reporters m_listeners;
        IStreamingReporterPtr m_reporter = nullptr;
        ReporterPreferences m_preferences;

    public:
        ListeningReporter();

        void addListener( IStreamingReporterPtr&& listener );
        void addReporter( IStreamingReporterPtr&& reporter );

    public: // IStreamingReporter

        ReporterPreferences getPreferences() const override;

        void noMatchingTestCases( std::string const& spec ) override;

        void reportInvalidArguments(std::string const&arg) override;

        static std::set<Verbosity> getSupportedVerbosities();

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
        void benchmarkPreparing(std::string const& name) override;
        void benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) override;
        void benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) override;
        void benchmarkFailed(std::string const&) override;
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

        void testRunStarting( TestRunInfo const& testRunInfo ) override;
        void testGroupStarting( GroupInfo const& groupInfo ) override;
        void testCaseStarting( TestCaseInfo const& testInfo ) override;
        void sectionStarting( SectionInfo const& sectionInfo ) override;
        void assertionStarting( AssertionInfo const& assertionInfo ) override;

        // The return value indicates if the messages buffer should be cleared:
        bool assertionEnded( AssertionStats const& assertionStats ) override;
        void sectionEnded( SectionStats const& sectionStats ) override;
        void testCaseEnded( TestCaseStats const& testCaseStats ) override;
        void testGroupEnded( TestGroupStats const& testGroupStats ) override;
        void testRunEnded( TestRunStats const& testRunStats ) override;

        void skipTest( TestCaseInfo const& testInfo ) override;
        bool isMulti() const override;

    };

} // end namespace Catch

// end catch_reporter_listening.h
namespace Catch {

    ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig )
    :   m_stream( &_fullConfig->stream() ), m_fullConfig( _fullConfig ) {}

    ReporterConfig::ReporterConfig( IConfigPtr const& _fullConfig, std::ostream& _stream )
    :   m_stream( &_stream ), m_fullConfig( _fullConfig ) {}

    std::ostream& ReporterConfig::stream() const { return *m_stream; }
    IConfigPtr ReporterConfig::fullConfig() const { return m_fullConfig; }

    TestRunInfo::TestRunInfo( std::string const& _name ) : name( _name ) {}

    GroupInfo::GroupInfo(  std::string const& _name,
                           std::size_t _groupIndex,
                           std::size_t _groupsCount )
    :   name( _name ),
        groupIndex( _groupIndex ),
        groupsCounts( _groupsCount )
    {}

     AssertionStats::AssertionStats( AssertionResult const& _assertionResult,
                                     std::vector<MessageInfo> const& _infoMessages,
                                     Totals const& _totals )
    :   assertionResult( _assertionResult ),
        infoMessages( _infoMessages ),
        totals( _totals )
    {
        assertionResult.m_resultData.lazyExpression.m_transientExpression =
_assertionResult.m_resultData.lazyExpression.m_transientExpression;

        if( assertionResult.hasMessage() ) {
            // Copy message into messages list.
            // !TBD This should have been done earlier, somewhere
            MessageBuilder builder( assertionResult.getTestMacroName(),
assertionResult.getSourceInfo(), assertionResult.getResultType() );
            builder « assertionResult.getMessage();
            builder.m_info.message = builder.m_stream.str();

            infoMessages.push_back( builder.m_info );
        }
    }

     AssertionStats::~AssertionStats() = default;

    SectionStats::SectionStats(  SectionInfo const& _sectionInfo,
                                 Counts const& _assertions,
                                 double _durationInSeconds,
                                 bool _missingAssertions )
    :   sectionInfo( _sectionInfo ),
        assertions( _assertions ),
        durationInSeconds( _durationInSeconds ),
        missingAssertions( _missingAssertions )
    {}

    SectionStats::~SectionStats() = default;

    TestCaseStats::TestCaseStats(  TestCaseInfo const& _testInfo,
                                   Totals const& _totals,
                                   std::string const& _stdOut,
                                   std::string const& _stdErr,
                                   bool _aborting )
    : testInfo( _testInfo ),
        totals( _totals ),
        stdOut( _stdOut ),
        stdErr( _stdErr ),
        aborting( _aborting )
    {}

    TestCaseStats::~TestCaseStats() = default;

    TestGroupStats::TestGroupStats( GroupInfo const& _groupInfo,
                                    Totals const& _totals,
                                    bool _aborting )
    :   groupInfo( _groupInfo ),
        totals( _totals ),
        aborting( _aborting )
    {}

    TestGroupStats::TestGroupStats( GroupInfo const& _groupInfo )
    :   groupInfo( _groupInfo ),
        aborting( false )
    {}

    TestGroupStats::~TestGroupStats() = default;

    TestRunStats::TestRunStats(   TestRunInfo const& _runInfo,
                    Totals const& _totals,
                    bool _aborting )
    :   runInfo( _runInfo ),
        totals( _totals ),
        aborting( _aborting )
    {}

    TestRunStats::~TestRunStats() = default;

    void IStreamingReporter::fatalErrorEncountered( StringRef ) {}
    bool IStreamingReporter::isMulti() const { return false; }

    IReporterFactory::~IReporterFactory() = default;
    IReporterRegistry::~IReporterRegistry() = default;

} // end namespace Catch
// end catch_interfaces_reporter.cpp
// start catch_interfaces_runner.cpp

namespace Catch {
    IRunner::~IRunner() = default;
}
// end catch_interfaces_runner.cpp
// start catch_interfaces_testcase.cpp

namespace Catch {
    ITestInvoker::~ITestInvoker() = default;
    ITestCaseRegistry::~ITestCaseRegistry() = default;
}
// end catch_interfaces_testcase.cpp
// start catch_leak_detector.cpp

#ifdef CATCH_CONFIG_WINDOWS_CRTDBG
#include <crtdbg.h>

namespace Catch {

    LeakDetector::LeakDetector() {
        int flag = _CrtSetDbgFlag(_CRTDBG_REPORT_FLAG);
        flag |= _CRTDBG_LEAK_CHECK_DF;
        flag |= _CRTDBG_ALLOC_MEM_DF;
        _CrtSetDbgFlag(flag);
        _CrtSetReportMode(_CRT_WARN, _CRTDBG_MODE_FILE | _CRTDBG_MODE_DEBUG);
        _CrtSetReportFile(_CRT_WARN, _CRTDBG_FILE_STDERR);
        // Change this to leaking allocation's number to break there
        _CrtSetBreakAlloc(-1);
    }
}

#else

    Catch::LeakDetector::LeakDetector() {}

#endif

Catch::LeakDetector::~LeakDetector() {
    Catch::cleanUp();
}
// end catch_leak_detector.cpp
// start catch_list.cpp

// start catch_list.h

#include <set>

namespace Catch {

    std::size_t listTests( Config const& config );

    std::size_t listTestsNamesOnly( Config const& config );

    struct TagInfo {
        void add( std::string const& spelling );
        std::string all() const;

        std::set<std::string> spellings;
        std::size_t count = 0;
    };

    std::size_t listTags( Config const& config );

    std::size_t listReporters();

    Option<std::size_t> list( std::shared_ptr<Config> const& config );

} // end namespace Catch

// end catch_list.h
// start catch_text.h

namespace Catch {
    using namespace clara::TextFlow;
}

// end catch_text.h
#include <limits>
#include <algorithm>
#include <iomanip>

```

```
namespace Catch {

    std::size_t listTests( Config const& config ) {
        TestSpec const& testSpec = config.testSpec();
        if( config.hasTestFilters() )
            Catch::cout() « "Matching test cases:\n";
        else {
            Catch::cout() « "All available test cases:\n";
        }

        auto matchedTestCases = filterTests( getAllTestCasesSorted( config ), testSpec, config );
        for( auto const& testCaseInfo : matchedTestCases ) {
            Colour::Code colour = testCaseInfo.isHidden()
                ? Colour::SecondaryText
                : Colour::None;
            Colour colourGuard( colour );

            Catch::cout() « Column( testCaseInfo.name ).initialIndent( 2 ).indent( 4 ) « "\n";
            if( config.verbosity() >= Verbosity::High ) {
                Catch::cout() « Column( Catch::Detail::stringify( testCaseInfo.lineInfo ) ).indent(4)
« std::endl;
                std::string description = testCaseInfo.description;
                if( description.empty() )
                    description = "(NO DESCRIPTION)";
                Catch::cout() « Column( description ).indent(4) « std::endl;
            }
            if( !testCaseInfo.tags.empty() )
                Catch::cout() « Column( testCaseInfo.tagsAsString() ).indent( 6 ) « "\n";
        }

        if( !config.hasTestFilters() )
            Catch::cout() « pluralise( matchedTestCases.size(), "test case" ) « '\n' « std::endl;
        else
            Catch::cout() « pluralise( matchedTestCases.size(), "matching test case" ) « '\n' «
std::endl;
        return matchedTestCases.size();
    }

    std::size_t listTestsNamesOnly( Config const& config ) {
        TestSpec const& testSpec = config.testSpec();
        std::size_t matchedTests = 0;
        std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
testSpec, config );
        for( auto const& testCaseInfo : matchedTestCases ) {
            matchedTests++;
            if( startsWith( testCaseInfo.name, '#' ) )
               Catch::cout() « '"' « testCaseInfo.name « '"';
            else
               Catch::cout() « testCaseInfo.name;
            if ( config.verbosity() >= Verbosity::High )
                Catch::cout() « "\t@" « testCaseInfo.lineInfo;
            Catch::cout() « std::endl;
        }
        return matchedTests;
    }

    void TagInfo::add( std::string const& spelling ) {
        ++count;
        spellings.insert( spelling );
    }

    std::string TagInfo::all() const {
        size_t size = 0;
        for (auto const& spelling : spellings) {
            // Add 2 for the brackes
            size += spelling.size() + 2;
        }

        std::string out; out.reserve(size);
        for (auto const& spelling : spellings) {
            out += '[';
            out += spelling;
            out += ']';
        }
        return out;
    }

    std::size_t listTags( Config const& config ) {
        TestSpec const& testSpec = config.testSpec();
        if( config.hasTestFilters() )
            Catch::cout() « "Tags for matching test cases:\n";
        else {
            Catch::cout() « "All available tags:\n";
        }

        std::map<std::string, TagInfo> tagCounts;

        std::vector<TestCase> matchedTestCases = filterTests( getAllTestCasesSorted( config ),
testSpec, config );
        for( auto const& testCase : matchedTestCases ) {
            for( auto const& tagName : testCase.getTestCaseInfo().tags ) {
                std::string lcaseTagName = toLower( tagName );
                auto countIt = tagCounts.find( lcaseTagName );
                if( countIt == tagCounts.end() )
                    countIt = tagCounts.insert( std::make_pair( lcaseTagName, TagInfo() ) ).first;
                countIt->second.add( tagName );
            }
        }

        for( auto const& tagCount : tagCounts ) {
            ReusableStringStream rss;
            rss « "  " « std::setw(2) « tagCount.second.count « "  ";
            auto str = rss.str();
            auto wrapper = Column( tagCount.second.all() )
                                                    .initialIndent( 0 )
                                                    .indent( str.size() )
                                                    .width( CATCH_CONFIG_CONSOLE_WIDTH-10 );
            Catch::cout() « str « wrapper « '\n';
        }
        Catch::cout() « pluralise( tagCounts.size(), "tag" ) « '\n' « std::endl;
        return tagCounts.size();
    }

    std::size_t listReporters() {
        Catch::cout() « "Available reporters:\n";
        IReporterRegistry::FactoryMap const& factories =
getRegistryHub().getReporterRegistry().getFactories();
        std::size_t maxNameLen = 0;
        for( auto const& factoryKvp : factories )
            maxNameLen = (std::max)( maxNameLen, factoryKvp.first.size() );

        for( auto const& factoryKvp : factories ) {
            Catch::cout()
                    « Column( factoryKvp.first + ":" )
                            .indent(2)
                            .width( 5+maxNameLen )
                    +  Column( factoryKvp.second->getDescription() )
                            .initialIndent(0)
                            .indent(2)
                            .width( CATCH_CONFIG_CONSOLE_WIDTH - maxNameLen-8 )
                    « "\n";
        }
        Catch::cout() « std::endl;
        return factories.size();
    }

    Option<std::size_t> list( std::shared_ptr<Config> const& config ) {
        Option<std::size_t> listedCount;
        getCurrentMutableContext().setConfig( config );
        if( config->listTests() )
            listedCount = listedCount.valueOr(0) + listTests( *config );
        if( config->listTestNamesOnly() )
            listedCount = listedCount.valueOr(0) + listTestsNamesOnly( *config );
        if( config->listTags() )
            listedCount = listedCount.valueOr(0) + listTags( *config );
        if( config->listReporters() )
            listedCount = listedCount.valueOr(0) + listReporters();
        return listedCount;
    }

} // end namespace Catch
// end catch_list.cpp
// start catch_matchers.cpp

namespace Catch {
namespace Matchers {
    namespace Impl {

        std::string MatcherUntypedBase::toString() const {
            if( m_cachedToString.empty() )
                m_cachedToString = describe();
            return m_cachedToString;
        }

        MatcherUntypedBase::~MatcherUntypedBase() = default;

    } // namespace Impl
} // namespace Matchers

using namespace Matchers;
using Matchers::Impl::MatcherBase;

} // namespace Catch
// end catch_matchers.cpp
// start catch_matchers_exception.cpp

namespace Catch {
namespace Matchers {
namespace Exception {

bool ExceptionMessageMatcher::match(std::exception const& ex) const {
    return ex.what() == m_message;
}

std::string ExceptionMessageMatcher::describe() const {
    return "exception message matches \"" + m_message + "\"";
}

}
Exception::ExceptionMessageMatcher Message(std::string const& message) {
    return Exception::ExceptionMessageMatcher(message);
}

// namespace Exception
} // namespace Matchers
} // namespace Catch
// end catch_matchers_exception.cpp
// start catch_matchers_floating.cpp

// start catch_polyfills.hpp

namespace Catch {
    bool isnan(float f);
    bool isnan(double d);
}

// end catch_polyfills.hpp
// start catch_to_string.hpp

#include <string>

namespace Catch {
    template <typename T>
    std::string to_string(T const& t) {
#if defined(CATCH_CONFIG_CPP11_TO_STRING)
        return std::to_string(t);
#else
        ReusableStringStream rss;
        rss « t;
        return rss.str();
#endif
    }
} // end namespace Catch

// end catch_to_string.hpp
#include <algorithm>
#include <cmath>
#include <cstdlib>
#include <cstdint>
#include <cstring>
#include <sstream>
#include <type_traits>
#include <iomanip>
#include <limits>

namespace Catch {
namespace {

    int32_t convert(float f) {
        static_assert(sizeof(float) == sizeof(int32_t), "Important ULP matcher assumption violated");
        int32_t i;
        std::memcpy(&i, &f, sizeof(f));
        return i;
    }

    int64_t convert(double d) {
        static_assert(sizeof(double) == sizeof(int64_t), "Important ULP matcher assumption violated");
        int64_t i;
        std::memcpy(&i, &d, sizeof(d));
        return i;
    }

    template <typename FP>
    bool almostEqualUlps(FP lhs, FP rhs, uint64_t maxUlpDiff) {
        // Comparison with NaN should always be false.
        // This way we can rule it out before getting into the ugly details
        if (Catch::isnan(lhs) || Catch::isnan(rhs)) {
            return false;
        }

        auto lc = convert(lhs);
        auto rc = convert(rhs);

        if ((lc < 0) != (rc < 0)) {
            // Potentially we can have +0 and -0
            return lhs == rhs;
        }

        // static cast as a workaround for IBM XLC
        auto ulpDiff = std::abs(static_cast<FP>(lc - rc));
        return static_cast<uint64_t>(ulpDiff) <= maxUlpDiff;
    }

#if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)

    float nextafter(float x, float y) {
        return ::nextafterf(x, y);
    }

    double nextafter(double x, double y) {
        return ::nextafter(x, y);
    }

#endif // ^^^ CATCH_CONFIG_GLOBAL_NEXTAFTER ^^^

template <typename FP>
FP step(FP start, FP direction, uint64_t steps) {
    for (uint64_t i = 0; i < steps; ++i) {
#if defined(CATCH_CONFIG_GLOBAL_NEXTAFTER)
        start = Catch::nextafter(start, direction);
#else
        start = std::nextafter(start, direction);
#endif
    }
    return start;
}

// Performs equivalent check of std::fabs(lhs - rhs) <= margin
// But without the subtraction to allow for INFINITY in comparison
bool marginComparison(double lhs, double rhs, double margin) {
    return (lhs + margin >= rhs) && (rhs + margin >= lhs);
}

template <typename FloatingPoint>
void write(std::ostream& out, FloatingPoint num) {
    out « std::scientific
        « std::setprecision(std::numeric_limits<FloatingPoint>::max_digits10 - 1)
        « num;
}

} // end anonymous namespace

namespace Matchers {
namespace Floating {

    enum class FloatingPointKind : uint8_t {
        Float,
        Double
    };

    WithinAbsMatcher::WithinAbsMatcher(double target, double margin)
        :m_target{ target }, m_margin{ margin } {
        CATCH_ENFORCE(margin >= 0, "Invalid margin: " « margin « '.'
            « " Margin has to be non-negative.");
    }

    // Performs equivalent check of std::fabs(lhs - rhs) <= margin
    // But without the subtraction to allow for INFINITY in comparison
    bool WithinAbsMatcher::match(double const& matchee) const {
        return (matchee + m_margin >= m_target) && (m_target + m_margin >= matchee);
    }

    std::string WithinAbsMatcher::describe() const {
        return "is within " + ::Catch::Detail::stringify(m_margin) + " of " +
::Catch::Detail::stringify(m_target);
    }

    WithinUlpsMatcher::WithinUlpsMatcher(double target, uint64_t ulps, FloatingPointKind baseType)
        :m_target{ target }, m_ulps{ ulps }, m_type{ baseType } {
        CATCH_ENFORCE(m_type == FloatingPointKind::Double
                   || m_ulps < (std::numeric_limits<uint32_t>::max)(),
            "Provided ULP is impossibly large for a float comparison.");
    }

#if defined(__clang__)
#pragma clang diagnostic push
// Clang <3.5 reports on the default branch in the switch below
#pragma clang diagnostic ignored "-Wunreachable-code"
```

```
#endif

    bool WithinUlpsMatcher::match(double const& matchee) const {
        switch (m_type) {
        case FloatingPointKind::Float:
            return almostEqualUlps<float>(static_cast<float>(matchee), static_cast<float>(m_target),
m_ulps);
        case FloatingPointKind::Double:
            return almostEqualUlps<double>(matchee, m_target, m_ulps);
        default:
            CATCH_INTERNAL_ERROR( "Unknown FloatingPointKind value" );
        }
    }

#if defined(__clang__)
#pragma clang diagnostic pop
#endif

    std::string WithinUlpsMatcher::describe() const {
        std::stringstream ret;

        ret « "is within " « m_ulps « " ULPs of ";

        if (m_type == FloatingPointKind::Float) {
            write(ret, static_cast<float>(m_target));
            ret « 'f';
        } else {
            write(ret, m_target);
        }

        ret « " ([";
        if (m_type == FloatingPointKind::Double) {
            write(ret, step(m_target, static_cast<double>(-INFINITY), m_ulps));
            ret « ", ";
            write(ret, step(m_target, static_cast<double>( INFINITY), m_ulps));
        } else {
            // We have to cast INFINITY to float because of MinGW, see #1782
            write(ret, step(static_cast<float>(m_target), static_cast<float>(-INFINITY), m_ulps));
            ret « ", ";
            write(ret, step(static_cast<float>(m_target), static_cast<float>( INFINITY), m_ulps));
        }
        ret « "])";

        return ret.str();
    }

    WithinRelMatcher::WithinRelMatcher(double target, double epsilon):
        m_target(target),
        m_epsilon(epsilon){
        CATCH_ENFORCE(m_epsilon >= 0., "Relative comparison with epsilon <  0 does not make sense.");
        CATCH_ENFORCE(m_epsilon  < 1., "Relative comparison with epsilon >= 1 does not make sense.");
    }

    bool WithinRelMatcher::match(double const& matchee) const {
        const auto relMargin = m_epsilon * (std::max)(std::fabs(matchee), std::fabs(m_target));
        return marginComparison(matchee, m_target,
                                std::isinf(relMargin)? 0 : relMargin);
    }

    std::string WithinRelMatcher::describe() const {
        Catch::ReusableStringStream sstr;
        sstr « "and " « m_target « " are within " « m_epsilon * 100. « "% of each other";
        return sstr.str();
    }

}// namespace Floating

Floating::WithinUlpsMatcher WithinULP(double target, uint64_t maxUlpDiff) {
    return Floating::WithinUlpsMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Double);
}

Floating::WithinUlpsMatcher WithinULP(float target, uint64_t maxUlpDiff) {
    return Floating::WithinUlpsMatcher(target, maxUlpDiff, Floating::FloatingPointKind::Float);
}

Floating::WithinAbsMatcher WithinAbs(double target, double margin) {
    return Floating::WithinAbsMatcher(target, margin);
}

Floating::WithinRelMatcher WithinRel(double target, double eps) {
    return Floating::WithinRelMatcher(target, eps);
}

Floating::WithinRelMatcher WithinRel(double target) {
    return Floating::WithinRelMatcher(target, std::numeric_limits<double>::epsilon() * 100);
}

```

```
Floating::WithinRelMatcher WithinRel(float target, float eps) {
    return Floating::WithinRelMatcher(target, eps);
}

Floating::WithinRelMatcher WithinRel(float target) {
    return Floating::WithinRelMatcher(target, std::numeric_limits<float>::epsilon() * 100);
}

} // namespace Matchers
} // namespace Catch
// end catch_matchers_floating.cpp
// start catch_matchers_generic.cpp

std::string Catch::Matchers::Generic::Detail::finalizeDescription(const std::string& desc) {
    if (desc.empty()) {
        return "matches undescribed predicate";
    } else {
        return "matches predicate: \"" + desc + '"';
    }
}
// end catch_matchers_generic.cpp
// start catch_matchers_string.cpp

#include <regex>

namespace Catch {
namespace Matchers {

    namespace StdString {

        CasedString::CasedString( std::string const& str, CaseSensitive::Choice caseSensitivity )
        :   m_caseSensitivity( caseSensitivity ),
            m_str( adjustString( str ) )
        {}
        std::string CasedString::adjustString( std::string const& str ) const {
            return m_caseSensitivity == CaseSensitive::No
                   ? toLower( str )
                   : str;
        }
        std::string CasedString::caseSensitivitySuffix() const {
            return m_caseSensitivity == CaseSensitive::No
                   ? " (case insensitive)"
                   : std::string();
        }

        StringMatcherBase::StringMatcherBase( std::string const& operation, CasedString const&
comparator )
        : m_comparator( comparator ),
          m_operation( operation ) {
        }

        std::string StringMatcherBase::describe() const {
            std::string description;
            description.reserve(5 + m_operation.size() + m_comparator.m_str.size() +
                                        m_comparator.caseSensitivitySuffix().size());
            description += m_operation;
            description += ": \"";
            description += m_comparator.m_str;
            description += "\"";
            description += m_comparator.caseSensitivitySuffix();
            return description;
        }

        EqualsMatcher::EqualsMatcher( CasedString const& comparator ) : StringMatcherBase( "equals",
comparator ) {}

        bool EqualsMatcher::match( std::string const& source ) const {
            return m_comparator.adjustString( source ) == m_comparator.m_str;
        }

        ContainsMatcher::ContainsMatcher( CasedString const& comparator ) : StringMatcherBase(
"contains", comparator ) {}

        bool ContainsMatcher::match( std::string const& source ) const {
            return contains( m_comparator.adjustString( source ), m_comparator.m_str );
        }

        StartsWithMatcher::StartsWithMatcher( CasedString const& comparator ) : StringMatcherBase(
"starts with", comparator ) {}

        bool StartsWithMatcher::match( std::string const& source ) const {
            return startsWith( m_comparator.adjustString( source ), m_comparator.m_str );
        }

        EndsWithMatcher::EndsWithMatcher( CasedString const& comparator ) : StringMatcherBase( "ends
with", comparator ) {}

```

```
        bool EndsWithMatcher::match( std::string const& source ) const {
            return endsWith( m_comparator.adjustString( source ), m_comparator.m_str );
        }

        RegexMatcher::RegexMatcher(std::string regex, CaseSensitive::Choice caseSensitivity):
m_regex(std::move(regex)), m_caseSensitivity(caseSensitivity) {}

        bool RegexMatcher::match(std::string const& matchee) const {
            auto flags = std::regex::ECMAScript; // ECMAScript is the default syntax option anyway
            if (m_caseSensitivity == CaseSensitive::Choice::No) {
                flags |= std::regex::icase;
            }
            auto reg = std::regex(m_regex, flags);
            return std::regex_match(matchee, reg);
        }

        std::string RegexMatcher::describe() const {
            return "matches " + ::Catch::Detail::stringify(m_regex) + ((m_caseSensitivity ==
CaseSensitive::Choice::Yes)? " case sensitively" : " case insensitively");
        }

    } // namespace StdString

    StdString::EqualsMatcher Equals( std::string const& str, CaseSensitive::Choice caseSensitivity ) {
        return StdString::EqualsMatcher( StdString::CasedString( str, caseSensitivity) );
    }
    StdString::ContainsMatcher Contains( std::string const& str, CaseSensitive::Choice caseSensitivity
) {
        return StdString::ContainsMatcher( StdString::CasedString( str, caseSensitivity) );
    }
    StdString::EndsWithMatcher EndsWith( std::string const& str, CaseSensitive::Choice caseSensitivity
) {
        return StdString::EndsWithMatcher( StdString::CasedString( str, caseSensitivity) );
    }
    StdString::StartsWithMatcher StartsWith( std::string const& str, CaseSensitive::Choice
caseSensitivity ) {
        return StdString::StartsWithMatcher( StdString::CasedString( str, caseSensitivity) );
    }

    StdString::RegexMatcher Matches(std::string const& regex, CaseSensitive::Choice caseSensitivity) {
        return StdString::RegexMatcher(regex, caseSensitivity);
    }

} // namespace Matchers
} // namespace Catch
// end catch_matchers_string.cpp
// start catch_message.cpp

// start catch_uncaught_exceptions.h

namespace Catch {
    bool uncaught_exceptions();
} // end namespace Catch

// end catch_uncaught_exceptions.h
#include <cassert>
#include <stack>

namespace Catch {

    MessageInfo::MessageInfo(   StringRef const& _macroName,
                                SourceLineInfo const& _lineInfo,
                                ResultWas::OfType _type )
    :   macroName( _macroName ),
        lineInfo( _lineInfo ),
        type( _type ),
        sequence( ++globalCount )
    {}

    bool MessageInfo::operator==( MessageInfo const& other ) const {
        return sequence == other.sequence;
    }

    bool MessageInfo::operator<( MessageInfo const& other ) const {
        return sequence < other.sequence;
    }

    // This may need protecting if threading support is added
    unsigned int MessageInfo::globalCount = 0;


    Catch::MessageBuilder::MessageBuilder( StringRef const& macroName,
                                           SourceLineInfo const& lineInfo,
                                           ResultWas::OfType type )
        :m_info(macroName, lineInfo, type) {}


    ScopedMessage::ScopedMessage( MessageBuilder const& builder )
    : m_info( builder.m_info ), m_moved()
    {
        m_info.message = builder.m_stream.str();
        getResultCapture().pushScopedMessage( m_info );
    }

    ScopedMessage::ScopedMessage( ScopedMessage&& old )
    : m_info( old.m_info ), m_moved()
    {
        old.m_moved = true;
    }

    ScopedMessage::~ScopedMessage() {
        if ( !uncaught_exceptions() && !m_moved ){
            getResultCapture().popScopedMessage(m_info);
        }
    }

    Capturer::Capturer( StringRef macroName, SourceLineInfo const& lineInfo, ResultWas::OfType resultType, StringRef names ) {
        auto trimmed = [&] (size_t start, size_t end) {
            while (names[start] == ',' || isspace(static_cast<unsigned char>(names[start]))) {
                ++start;
            }
            while (names[end] == ',' || isspace(static_cast<unsigned char>(names[end]))) {
                --end;
            }
            return names.substr(start, end - start + 1);
        };
        auto skipq = [&] (size_t start, char quote) {
            for (auto i = start + 1; i < names.size() ; ++i) {
                if (names[i] == quote)
                    return i;
                if (names[i] == '\\')
                    ++i;
            }
            CATCH_INTERNAL_ERROR("CAPTURE parsing encountered unmatched quote");
        };

        size_t start = 0;
        std::stack<char> openings;
        for (size_t pos = 0; pos < names.size(); ++pos) {
            char c = names[pos];
            switch (c) {
            case '[':
            case '{':
            case '(':
            // It is basically impossible to disambiguate between
            // comparison and start of template args in this context
//            case '<':
                openings.push(c);
                break;
            case ']':
            case '}':
            case ')':
//           case '>':
                openings.pop();
                break;
            case '"':
            case '\'':
                pos = skipq(pos, c);
                break;
            case ',':
                if (start != pos && openings.empty()) {
                    m_messages.emplace_back(macroName, lineInfo, resultType);
                    m_messages.back().message = static_cast<std::string>(trimmed(start, pos));
                    m_messages.back().message += " := ";
                    start = pos;
                }
            }
        }
        assert(openings.empty() && "Mismatched openings");
        m_messages.emplace_back(macroName, lineInfo, resultType);
        m_messages.back().message = static_cast<std::string>(trimmed(start, names.size() - 1));
        m_messages.back().message += " := ";
    }
    Capturer::~Capturer() {
        if ( !uncaught_exceptions() ){
            assert( m_captured == m_messages.size() );
            for( size_t i = 0; i < m_captured; ++i  )
                m_resultCapture.popScopedMessage( m_messages[i] );
        }
    }

    void Capturer::captureValue( size_t index, std::string const& value ) {
        assert( index < m_messages.size() );
        m_messages[index].message += value;
        m_resultCapture.pushScopedMessage( m_messages[index] );
        m_captured++;
    }

} // end namespace Catch
// end catch_message.cpp
// start catch_output_redirect.cpp

// start catch_output_redirect.h
#ifndef TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
#define TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H

#include <cstdio>
#include <iosfwd>
#include <string>

namespace Catch {

    class RedirectedStream {
        std::ostream& m_originalStream;
        std::ostream& m_redirectionStream;
        std::streambuf* m_prevBuf;

    public:
        RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream );
        ~RedirectedStream();
    };

    class RedirectedStdOut {
        ReusableStringStream m_rss;
        RedirectedStream m_cout;
    public:
        RedirectedStdOut();
        auto str() const -> std::string;
    };

    // StdErr has two constituent streams in C++, std::cerr and std::clog
    // This means that we need to redirect 2 streams into 1 to keep proper
    // order of writes
    class RedirectedStdErr {
        ReusableStringStream m_rss;
        RedirectedStream m_cerr;
        RedirectedStream m_clog;
    public:
        RedirectedStdErr();
        auto str() const -> std::string;
    };

    class RedirectedStreams {
    public:
        RedirectedStreams(RedirectedStreams const&) = delete;
        RedirectedStreams& operator=(RedirectedStreams const&) = delete;
        RedirectedStreams(RedirectedStreams&&) = delete;
        RedirectedStreams& operator=(RedirectedStreams&&) = delete;

        RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr);
        ~RedirectedStreams();
    private:
        std::string& m_redirectedCout;
        std::string& m_redirectedCerr;
        RedirectedStdOut m_redirectedStdOut;
        RedirectedStdErr m_redirectedStdErr;
    };

#if defined(CATCH_CONFIG_NEW_CAPTURE)

    // Windows's implementation of std::tmpfile is terrible (it tries
    // to create a file inside system folder, thus requiring elevated
    // privileges for the binary), so we have to use tmpnam(_s) and
    // create the file ourselves there.
    class TempFile {
    public:
        TempFile(TempFile const&) = delete;
        TempFile& operator=(TempFile const&) = delete;
        TempFile(TempFile&&) = delete;
        TempFile& operator=(TempFile&&) = delete;

        TempFile();
        ~TempFile();

        std::FILE* getFile();
        std::string getContents();

    private:
        std::FILE* m_file = nullptr;
    #if defined(_MSC_VER)
```

```
        char m_buffer[L_tmpnam] = { 0 };
    #endif
    };

    class OutputRedirect {
    public:
        OutputRedirect(OutputRedirect const&) = delete;
        OutputRedirect& operator=(OutputRedirect const&) = delete;
        OutputRedirect(OutputRedirect&&) = delete;
        OutputRedirect& operator=(OutputRedirect&&) = delete;

        OutputRedirect(std::string& stdout_dest, std::string& stderr_dest);
        ~OutputRedirect();

    private:
        int m_originalStdout = -1;
        int m_originalStderr = -1;
        TempFile m_stdoutFile;
        TempFile m_stderrFile;
        std::string& m_stdoutDest;
        std::string& m_stderrDest;
    };

#endif

} // end namespace Catch

#endif // TWOBLUECUBES_CATCH_OUTPUT_REDIRECT_H
// end catch_output_redirect.h
#include <cstdio>
#include <cstring>
#include <fstream>
#include <sstream>
#include <stdexcept>

#if defined(CATCH_CONFIG_NEW_CAPTURE)
    #if defined(_MSC_VER)
    #include <io.h>      //_dup and _dup2
    #define dup _dup
    #define dup2 _dup2
    #define fileno _fileno
    #else
    #include <unistd.h>  // dup and dup2
    #endif
#endif

namespace Catch {

    RedirectedStream::RedirectedStream( std::ostream& originalStream, std::ostream& redirectionStream
)
    :   m_originalStream( originalStream ),
        m_redirectionStream( redirectionStream ),
        m_prevBuf( m_originalStream.rdbuf() )
    {
        m_originalStream.rdbuf( m_redirectionStream.rdbuf() );
    }

    RedirectedStream::~RedirectedStream() {
        m_originalStream.rdbuf( m_prevBuf );
    }

    RedirectedStdOut::RedirectedStdOut() : m_cout( Catch::cout(), m_rss.get() ) {}
    auto RedirectedStdOut::str() const -> std::string { return m_rss.str(); }

    RedirectedStdErr::RedirectedStdErr()
    :   m_cerr( Catch::cerr(), m_rss.get() ),
        m_clog( Catch::clog(), m_rss.get() )
    {}
    auto RedirectedStdErr::str() const -> std::string { return m_rss.str(); }

    RedirectedStreams::RedirectedStreams(std::string& redirectedCout, std::string& redirectedCerr)
    :   m_redirectedCout(redirectedCout),
        m_redirectedCerr(redirectedCerr)
    {}

    RedirectedStreams::~RedirectedStreams() {
        m_redirectedCout += m_redirectedStdOut.str();
        m_redirectedCerr += m_redirectedStdErr.str();
    }

#if defined(CATCH_CONFIG_NEW_CAPTURE)

#if defined(_MSC_VER)
    TempFile::TempFile() {
        if (tmpnam_s(m_buffer)) {
            CATCH_RUNTIME_ERROR("Could not get a temp filename");
        }
        if (fopen_s(&m_file, m_buffer, "w+")) {
            char buffer[100];
            if (strerror_s(buffer, errno)) {
                CATCH_RUNTIME_ERROR("Could not translate errno to a string");
            }
            CATCH_RUNTIME_ERROR("Could not open the temp file: '" « m_buffer « "' because: " «
buffer);
        }
    }
#else
    TempFile::TempFile() {
        m_file = std::tmpfile();
        if (!m_file) {
            CATCH_RUNTIME_ERROR("Could not create a temp file.");
        }
    }

#endif

    TempFile::~TempFile() {
         // TBD: What to do about errors here?
         std::fclose(m_file);
         // We manually create the file on Windows only, on Linux
         // it will be autodeleted
#if defined(_MSC_VER)
         std::remove(m_buffer);
#endif
    }

    FILE* TempFile::getFile() {
        return m_file;
    }

    std::string TempFile::getContents() {
        std::stringstream sstr;
        char buffer[100] = {};
        std::rewind(m_file);
        while (std::fgets(buffer, sizeof(buffer), m_file)) {
            sstr « buffer;
        }
        return sstr.str();
    }

    OutputRedirect::OutputRedirect(std::string& stdout_dest, std::string& stderr_dest) :
        m_originalStdout(dup(1)),
        m_originalStderr(dup(2)),
        m_stdoutDest(stdout_dest),
        m_stderrDest(stderr_dest) {
        dup2(fileno(m_stdoutFile.getFile()), 1);
        dup2(fileno(m_stderrFile.getFile()), 2);
    }

    OutputRedirect::~OutputRedirect() {
        Catch::cout() « std::flush;
        fflush(stdout);
        // Since we support overriding these streams, we flush cerr
        // even though std::cerr is unbuffered
        Catch::cerr() « std::flush;
        Catch::clog() « std::flush;
        fflush(stderr);

        dup2(m_originalStdout, 1);
        dup2(m_originalStderr, 2);

        m_stdoutDest += m_stdoutFile.getContents();
        m_stderrDest += m_stderrFile.getContents();
    }

#endif // CATCH_CONFIG_NEW_CAPTURE

} // namespace Catch

#if defined(CATCH_CONFIG_NEW_CAPTURE)
    #if defined(_MSC_VER)
    #undef dup
    #undef dup2
    #undef fileno
    #endif
#endif
// end catch_output_redirect.cpp
// start catch_polyfills.cpp

#include <cmath>

namespace Catch {

#if !defined(CATCH_CONFIG_POLYFILL_ISNAN)
```

```
    bool isnan(float f) {
        return std::isnan(f);
    }
    bool isnan(double d) {
        return std::isnan(d);
    }
#else
    // For now we only use this for embarcadero
    bool isnan(float f) {
        return std::_isnan(f);
    }
    bool isnan(double d) {
        return std::_isnan(d);
    }
#endif

} // end namespace Catch
// end catch_polyfills.cpp
// start catch_random_number_generator.cpp

namespace Catch {

namespace {

#if defined(_MSC_VER)
#pragma warning(push)
#pragma warning(disable:4146) // we negate uint32 during the rotate
#endif
        // Safe rotr implementation thanks to John Regehr
        uint32_t rotate_right(uint32_t val, uint32_t count) {
            const uint32_t mask = 31;
            count &= mask;
            return (val » count) | (val « (-count & mask));
        }

#if defined(_MSC_VER)
#pragma warning(pop)
#endif

}

    SimplePcg32::SimplePcg32(result_type seed_) {
        seed(seed_);
    }

    void SimplePcg32::seed(result_type seed_) {
        m_state = 0;
        (*this)();
        m_state += seed_;
        (*this)();
    }

    void SimplePcg32::discard(uint64_t skip) {
        // We could implement this to run in O(log n) steps, but this
        // should suffice for our use case.
        for (uint64_t s = 0; s < skip; ++s) {
            static_cast<void>((*this)());
        }
    }

    SimplePcg32::result_type SimplePcg32::operator()() {
        // prepare the output value
        const uint32_t xorshifted = static_cast<uint32_t>(((m_state » 18u) ^ m_state) » 27u);
        const auto output = rotate_right(xorshifted, m_state » 59u);

        // advance state
        m_state = m_state * 6364136223846793005ULL + s_inc;

        return output;
    }

    bool operator==(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
        return lhs.m_state == rhs.m_state;
    }

    bool operator!=(SimplePcg32 const& lhs, SimplePcg32 const& rhs) {
        return lhs.m_state != rhs.m_state;
    }
}
// end catch_random_number_generator.cpp
// start catch_registry_hub.cpp

// start catch_test_case_registry_impl.h

#include <vector>
#include <set>
#include <algorithm>
```

```
#include <ios>

namespace Catch {

    class TestCase;
    struct IConfig;

    std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
unsortedTestCases );

    bool isThrowSafe( TestCase const& testCase, IConfig const& config );
    bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config );

    void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions );

    std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config );
    std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config );

    class TestRegistry : public ITestCaseRegistry {
    public:
        virtual ~TestRegistry() = default;

        virtual void registerTest( TestCase const& testCase );

        std::vector<TestCase> const& getAllTests() const override;
        std::vector<TestCase> const& getAllTestsSorted( IConfig const& config ) const override;

    private:
        std::vector<TestCase> m_functions;
        mutable RunTests::InWhatOrder m_currentSortOrder = RunTests::InDeclarationOrder;
        mutable std::vector<TestCase> m_sortedFunctions;
        std::size_t m_unnamedCount = 0;
        std::ios_base::Init m_ostreamInit; // Forces cout/ cerr to be initialised
    };


    class TestInvokerAsFunction : public ITestInvoker {
        void(*m_testAsFunction)();
    public:
        TestInvokerAsFunction( void(*testAsFunction)() ) noexcept;

        void invoke() const override;
    };

    std::string extractClassName( StringRef const& classOrQualifiedMethodName );


} // end namespace Catch

// end catch_test_case_registry_impl.h
// start catch_reporter_registry.h

#include <map>

namespace Catch {

    class ReporterRegistry : public IReporterRegistry {

    public:

        ~ReporterRegistry() override;

        IStreamingReporterPtr create( std::string const& name, IConfigPtr const& config ) const
override;

        void registerReporter( std::string const& name, IReporterFactoryPtr const& factory );
        void registerListener( IReporterFactoryPtr const& factory );

        FactoryMap const& getFactories() const override;
        Listeners const& getListeners() const override;

    private:
        FactoryMap m_factories;
        Listeners m_listeners;
    };
}

// end catch_reporter_registry.h
// start catch_tag_alias_registry.h

// start catch_tag_alias.h

#include <string>

namespace Catch {

    struct TagAlias {
        TagAlias(std::string const& _tag, SourceLineInfo _lineInfo);

        std::string tag;
        SourceLineInfo lineInfo;
    };

} // end namespace Catch

// end catch_tag_alias.h
#include <map>

namespace Catch {

    class TagAliasRegistry : public ITagAliasRegistry {
    public:
        ~TagAliasRegistry() override;
        TagAlias const* find( std::string const& alias ) const override;
        std::string expandAliases( std::string const& unexpandedTestSpec ) const override;
        void add( std::string const& alias, std::string const& tag, SourceLineInfo const& lineInfo );

    private:
        std::map<std::string, TagAlias> m_registry;
    };

} // end namespace Catch

// end catch_tag_alias_registry.h
// start catch_startup_exception_registry.h

#include <vector>
#include <exception>

namespace Catch {

    class StartupExceptionRegistry {
#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
    public:
        void add(std::exception_ptr const& exception) noexcept;
        std::vector<std::exception_ptr> const& getExceptions() const noexcept;
    private:
        std::vector<std::exception_ptr> m_exceptions;
#endif
    };

} // end namespace Catch

// end catch_startup_exception_registry.h
// start catch_singletons.hpp

namespace Catch {

    struct ISingleton {
        virtual ~ISingleton();
    };

    void addSingleton( ISingleton* singleton );
    void cleanupSingletons();

    template<typename SingletonImplT, typename InterfaceT = SingletonImplT, typename MutableInterfaceT
= InterfaceT>
    class Singleton : SingletonImplT, public ISingleton {

        static auto getInternal() -> Singleton* {
            static Singleton* s_instance = nullptr;
            if( !s_instance ) {
                s_instance = new Singleton;
                addSingleton( s_instance );
            }
            return s_instance;
        }

    public:
        static auto get() -> InterfaceT const& {
            return *getInternal();
        }
        static auto getMutable() -> MutableInterfaceT& {
            return *getInternal();
        }
    };

} // namespace Catch

// end catch_singletons.hpp
namespace Catch {

    namespace {

        class RegistryHub : public IRegistryHub, public IMutableRegistryHub,
                            private NonCopyable {

        public: // IRegistryHub
            RegistryHub() = default;
            IReporterRegistry const& getReporterRegistry() const override {
                return m_reporterRegistry;
            }
            ITestCaseRegistry const& getTestCaseRegistry() const override {
                return m_testCaseRegistry;
            }
            IExceptionTranslatorRegistry const& getExceptionTranslatorRegistry() const override {
                return m_exceptionTranslatorRegistry;
            }
            ITagAliasRegistry const& getTagAliasRegistry() const override {
                return m_tagAliasRegistry;
            }
            StartupExceptionRegistry const& getStartupExceptionRegistry() const override {
                return m_exceptionRegistry;
            }

        public: // IMutableRegistryHub
            void registerReporter( std::string const& name, IReporterFactoryPtr const& factory )
override {
                m_reporterRegistry.registerReporter( name, factory );
            }
            void registerListener( IReporterFactoryPtr const& factory ) override {
                m_reporterRegistry.registerListener( factory );
            }
            void registerTest( TestCase const& testInfo ) override {
                m_testCaseRegistry.registerTest( testInfo );
            }
            void registerTranslator( const IExceptionTranslator* translator ) override {
                m_exceptionTranslatorRegistry.registerTranslator( translator );
            }
            void registerTagAlias( std::string const& alias, std::string const& tag, SourceLineInfo
const& lineInfo ) override {
                m_tagAliasRegistry.add( alias, tag, lineInfo );
            }
            void registerStartupException() noexcept override {
#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
                m_exceptionRegistry.add(std::current_exception());
#else
                CATCH_INTERNAL_ERROR("Attempted to register active exception under
CATCH_CONFIG_DISABLE_EXCEPTIONS!");
#endif
            }
            IMutableEnumValuesRegistry& getMutableEnumValuesRegistry() override {
                return m_enumValuesRegistry;
            }

        private:
            TestRegistry m_testCaseRegistry;
            ReporterRegistry m_reporterRegistry;
            ExceptionTranslatorRegistry m_exceptionTranslatorRegistry;
            TagAliasRegistry m_tagAliasRegistry;
            StartupExceptionRegistry m_exceptionRegistry;
            Detail::EnumValuesRegistry m_enumValuesRegistry;
        };
    }

    using RegistryHubSingleton = Singleton<RegistryHub, IRegistryHub, IMutableRegistryHub>;

    IRegistryHub const& getRegistryHub() {
        return RegistryHubSingleton::get();
    }
    IMutableRegistryHub& getMutableRegistryHub() {
        return RegistryHubSingleton::getMutable();
    }
    void cleanUp() {
        cleanupSingletons();
        cleanUpContext();
    }
    std::string translateActiveException() {
        return getRegistryHub().getExceptionTranslatorRegistry().translateActiveException();
    }

} // end namespace Catch
// end catch_registry_hub.cpp
// start catch_reporter_registry.cpp

namespace Catch {

    ReporterRegistry::~ReporterRegistry() = default;

    IStreamingReporterPtr ReporterRegistry::create( std::string const& name, IConfigPtr const& config
) const {
        auto it =  m_factories.find( name );
        if( it == m_factories.end() )
            return nullptr;
        return it->second->create( ReporterConfig( config ) );
    }

    void ReporterRegistry::registerReporter( std::string const& name, IReporterFactoryPtr const& factory ) {
        m_factories.emplace(name, factory);
    }
    void ReporterRegistry::registerListener( IReporterFactoryPtr const& factory ) {
        m_listeners.push_back( factory );
    }

    IReporterRegistry::FactoryMap const& ReporterRegistry::getFactories() const {
        return m_factories;
    }
    IReporterRegistry::Listeners const& ReporterRegistry::getListeners() const {
        return m_listeners;
    }

}
// end catch_reporter_registry.cpp
// start catch_result_type.cpp

namespace Catch {

    bool isOk( ResultWas::OfType resultType ) {
        return ( resultType & ResultWas::FailureBit ) == 0;
    }
    bool isJustInfo( int flags ) {
        return flags == ResultWas::Info;
    }

    ResultDisposition::Flags operator | ( ResultDisposition::Flags lhs, ResultDisposition::Flags rhs ) {
        return static_cast<ResultDisposition::Flags>( static_cast<int>( lhs ) | static_cast<int>( rhs ) );
    }

    bool shouldContinueOnFailure( int flags )    { return ( flags & ResultDisposition::ContinueOnFailure ) != 0; }
    bool shouldSuppressFailure( int flags )      { return ( flags & ResultDisposition::SuppressFail ) != 0; }

} // end namespace Catch
// end catch_result_type.cpp
// start catch_run_context.cpp

#include <cassert>
#include <algorithm>
#include <sstream>

namespace Catch {

    namespace Generators {
        struct GeneratorTracker : TestCaseTracking::TrackerBase, IGeneratorTracker {
            GeneratorBasePtr m_generator;

            GeneratorTracker( TestCaseTracking::NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent )
            :   TrackerBase( nameAndLocation, ctx, parent )
            {}
            ~GeneratorTracker();

            static GeneratorTracker& acquire( TrackerContext& ctx, TestCaseTracking::NameAndLocation const& nameAndLocation ) {
                std::shared_ptr<GeneratorTracker> tracker;

                ITracker& currentTracker = ctx.currentTracker();
                // Under specific circumstances, the generator we want
                // to acquire is also the current tracker. If this is
                // the case, we have to avoid looking through current
                // tracker's children, and instead return the current
                // tracker.
                // A case where this check is important is e.g.
                //     for (int i = 0; i < 5; ++i) {
                //         int n = GENERATE(1, 2);
                //     }
                //
                // without it, the code above creates 5 nested generators.
                if (currentTracker.nameAndLocation() == nameAndLocation) {
                    auto thisTracker = currentTracker.parent().findChild(nameAndLocation);
                    assert(thisTracker);
                    assert(thisTracker->isGeneratorTracker());
                    tracker = std::static_pointer_cast<GeneratorTracker>(thisTracker);
                } else if ( TestCaseTracking::ITrackerPtr childTracker = currentTracker.findChild(
nameAndLocation ) ) {
                    assert( childTracker );
                    assert( childTracker->isGeneratorTracker() );
                    tracker = std::static_pointer_cast<GeneratorTracker>( childTracker );
                } else {
                    tracker = std::make_shared<GeneratorTracker>( nameAndLocation, ctx,
&currentTracker );
                    currentTracker.addChild( tracker );
                }

                if( !tracker->isComplete() ) {
                    tracker->open();
                }

                return *tracker;
            }

            // TrackerBase interface
            bool isGeneratorTracker() const override { return true; }
            auto hasGenerator() const -> bool override {
                return !!m_generator;
            }
            void close() override {
                TrackerBase::close();
                // If a generator has a child (it is followed by a section)
                // and none of its children have started, then we must wait
                // until later to start consuming its values.
                // This catches cases where `GENERATE` is placed between two
                // `SECTION`s.
                // **The check for m_children.empty cannot be removed**.
                // doing so would break `GENERATE` _not_ followed by `SECTION`s.
                const bool should_wait_for_child = [&]() {
                    // No children -> nobody to wait for
                    if ( m_children.empty() ) {
                        return false;
                    }
                    // If at least one child started executing, don't wait
                    if ( std::find_if(
                             m_children.begin(),
                             m_children.end(),
                             []( TestCaseTracking::ITrackerPtr tracker ) {
                                 return tracker->hasStarted();
                             } ) != m_children.end() ) {
                        return false;
                    }

                    // No children have started. We need to check if they _can_
                    // start, and thus we should wait for them, or they cannot
                    // start (due to filters), and we shouldn't wait for them
                    auto* parent = m_parent;
                    // This is safe: there is always at least one section
                    // tracker in a test case tracking tree
                    while ( !parent->isSectionTracker() ) {
                        parent = &( parent->parent() );
                    }
                    assert( parent &&
                            "Missing root (test case) level section" );

                    auto const& parentSection =
                        static_cast<SectionTracker&>( *parent );
                    auto const& filters = parentSection.getFilters();
                    // No filters -> no restrictions on running sections
                    if ( filters.empty() ) {
                        return true;
                    }

                    for ( auto const& child : m_children ) {
                        if ( child->isSectionTracker() &&
                             std::find( filters.begin(),
                                        filters.end(),
                                        static_cast<SectionTracker&>( *child )
                                            .trimmedName() ) !=
                                 filters.end() ) {
                            return true;
                        }
                    }
                    return false;
                }();

                // This check is a bit tricky, because m_generator->next()
                // has a side-effect, where it consumes generator's current
                // value, but we do not want to invoke the side-effect if
                // this generator is still waiting for any child to start.
                if ( should_wait_for_child ||
                     ( m_runState == CompletedSuccessfully &&
                       m_generator->next() ) ) {
```

```
                    m_children.clear();
                    m_runState = Executing;
                }
            }

            // IGeneratorTracker interface
            auto getGenerator() const -> GeneratorBasePtr const& override {
                return m_generator;
            }
            void setGenerator( GeneratorBasePtr&& generator ) override {
                m_generator = std::move( generator );
            }
        };
        GeneratorTracker::~GeneratorTracker() {}
    }

    RunContext::RunContext(IConfigPtr const& _config, IStreamingReporterPtr&& reporter)
    :   m_runInfo(_config->name()),
        m_context(getCurrentMutableContext()),
        m_config(_config),
        m_reporter(std::move(reporter)),
        m_lastAssertionInfo{ StringRef(), SourceLineInfo("",0), StringRef(), ResultDisposition::Normal
},
        m_includeSuccessfulResults( m_config->includeSuccessfulResults() ||
m_reporter->getPreferences().shouldReportAllAssertions )
    {
        m_context.setRunner(this);
        m_context.setConfig(m_config);
        m_context.setResultCapture(this);
        m_reporter->testRunStarting(m_runInfo);
    }

    RunContext::~RunContext() {
        m_reporter->testRunEnded(TestRunStats(m_runInfo, m_totals, aborting()));
    }

    void RunContext::testGroupStarting(std::string const& testSpec, std::size_t groupIndex,
std::size_t groupsCount) {
        m_reporter->testGroupStarting(GroupInfo(testSpec, groupIndex, groupsCount));
    }

    void RunContext::testGroupEnded(std::string const& testSpec, Totals const& totals, std::size_t
groupIndex, std::size_t groupsCount) {
        m_reporter->testGroupEnded(TestGroupStats(GroupInfo(testSpec, groupIndex, groupsCount),
totals, aborting()));
    }

    Totals RunContext::runTest(TestCase const& testCase) {
        Totals prevTotals = m_totals;

        std::string redirectedCout;
        std::string redirectedCerr;

        auto const& testInfo = testCase.getTestCaseInfo();

        m_reporter->testCaseStarting(testInfo);

        m_activeTestCase = &testCase;

        ITracker& rootTracker = m_trackerContext.startRun();
        assert(rootTracker.isSectionTracker());
        static_cast<SectionTracker&>(rootTracker).addInitialFilters(m_config->getSectionsToRun());
        do {
            m_trackerContext.startCycle();
            m_testCaseTracker = &SectionTracker::acquire(m_trackerContext,
TestCaseTracking::NameAndLocation(testInfo.name, testInfo.lineInfo));
            runCurrentTest(redirectedCout, redirectedCerr);
        } while (!m_testCaseTracker->isSuccessfullyCompleted() && !aborting());

        Totals deltaTotals = m_totals.delta(prevTotals);
        if (testInfo.expectedToFail() && deltaTotals.testCases.passed > 0) {
            deltaTotals.assertions.failed++;
            deltaTotals.testCases.passed--;
            deltaTotals.testCases.failed++;
        }
        m_totals.testCases += deltaTotals.testCases;
        m_reporter->testCaseEnded(TestCaseStats(testInfo,
                                  deltaTotals,
                                  redirectedCout,
                                  redirectedCerr,
                                  aborting()));

        m_activeTestCase = nullptr;
        m_testCaseTracker = nullptr;

        return deltaTotals;
    }
```

```cpp

    IConfigPtr RunContext::config() const {
        return m_config;
    }

    IStreamingReporter& RunContext::reporter() const {
        return *m_reporter;
    }

    void RunContext::assertionEnded(AssertionResult const & result) {
        if (result.getResultType() == ResultWas::Ok) {
            m_totals.assertions.passed++;
            m_lastAssertionPassed = true;
        } else if (!result.isOk()) {
            m_lastAssertionPassed = false;
            if( m_activeTestCase->getTestCaseInfo().okToFail() )
                m_totals.assertions.failedButOk++;
            else
                m_totals.assertions.failed++;
        }
        else {
            m_lastAssertionPassed = true;
        }

        // We have no use for the return value (whether messages should be cleared), because messages
were made scoped
        // and should be let to clear themselves out.
        static_cast<void>(m_reporter->assertionEnded(AssertionStats(result, m_messages, m_totals)));

        if (result.getResultType() != ResultWas::Warning)
            m_messageScopes.clear();

        // Reset working state
        resetAssertionInfo();
        m_lastResult = result;
    }
    void RunContext::resetAssertionInfo() {
        m_lastAssertionInfo.macroName = StringRef();
        m_lastAssertionInfo.capturedExpression = "{Unknown expression after the reported line}"_sr;
    }

    bool RunContext::sectionStarted(SectionInfo const & sectionInfo, Counts & assertions) {
        ITracker& sectionTracker = SectionTracker::acquire(m_trackerContext,
TestCaseTracking::NameAndLocation(sectionInfo.name, sectionInfo.lineInfo));
        if (!sectionTracker.isOpen())
            return false;
        m_activeSections.push_back(&sectionTracker);

        m_lastAssertionInfo.lineInfo = sectionInfo.lineInfo;

        m_reporter->sectionStarting(sectionInfo);

        assertions = m_totals.assertions;

        return true;
    }
    auto RunContext::acquireGeneratorTracker( StringRef generatorName, SourceLineInfo const& lineInfo
) -> IGeneratorTracker& {
        using namespace Generators;
        GeneratorTracker& tracker = GeneratorTracker::acquire(m_trackerContext,
                                                              TestCaseTracking::NameAndLocation(
static_cast<std::string>(generatorName), lineInfo ) );
        m_lastAssertionInfo.lineInfo = lineInfo;
        return tracker;
    }

    bool RunContext::testForMissingAssertions(Counts& assertions) {
        if (assertions.total() != 0)
            return false;
        if (!m_config->warnAboutMissingAssertions())
            return false;
        if (m_trackerContext.currentTracker().hasChildren())
            return false;
        m_totals.assertions.failed++;
        assertions.failed++;
        return true;
    }

    void RunContext::sectionEnded(SectionEndInfo const & endInfo) {
        Counts assertions = m_totals.assertions - endInfo.prevAssertions;
        bool missingAssertions = testForMissingAssertions(assertions);

        if (!m_activeSections.empty()) {
            m_activeSections.back()->close();
            m_activeSections.pop_back();
        }

        m_reporter->sectionEnded(SectionStats(endInfo.sectionInfo, assertions,
endInfo.durationInSeconds, missingAssertions));
        m_messages.clear();
        m_messageScopes.clear();
    }

    void RunContext::sectionEndedEarly(SectionEndInfo const & endInfo) {
        if (m_unfinishedSections.empty())
            m_activeSections.back()->fail();
        else
            m_activeSections.back()->close();
        m_activeSections.pop_back();

        m_unfinishedSections.push_back(endInfo);
    }

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
    void RunContext::benchmarkPreparing(std::string const& name) {
        m_reporter->benchmarkPreparing(name);
    }
    void RunContext::benchmarkStarting( BenchmarkInfo const& info ) {
        m_reporter->benchmarkStarting( info );
    }
    void RunContext::benchmarkEnded( BenchmarkStats<> const& stats ) {
        m_reporter->benchmarkEnded( stats );
    }
    void RunContext::benchmarkFailed(std::string const & error) {
        m_reporter->benchmarkFailed(error);
    }
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

    void RunContext::pushScopedMessage(MessageInfo const & message) {
        m_messages.push_back(message);
    }

    void RunContext::popScopedMessage(MessageInfo const & message) {
        m_messages.erase(std::remove(m_messages.begin(), m_messages.end(), message),
m_messages.end());
    }

    void RunContext::emplaceUnscopedMessage( MessageBuilder const& builder ) {
        m_messageScopes.emplace_back( builder );
    }

    std::string RunContext::getCurrentTestName() const {
        return m_activeTestCase
            ? m_activeTestCase->getTestCaseInfo().name
            : std::string();
    }

    const AssertionResult * RunContext::getLastResult() const {
        return &(*m_lastResult);
    }

    void RunContext::exceptionEarlyReported() {
        m_shouldReportUnexpected = false;
    }

    void RunContext::handleFatalErrorCondition( StringRef message ) {
        // First notify reporter that bad things happened
        m_reporter->fatalErrorEncountered(message);

        // Don't rebuild the result -- the stringification itself can cause more fatal errors
        // Instead, fake a result data.
        AssertionResultData tempResult( ResultWas::FatalErrorCondition, { false } );
        tempResult.message = static_cast<std::string>(message);
        AssertionResult result(m_lastAssertionInfo, tempResult);

        assertionEnded(result);

        handleUnfinishedSections();

        // Recreate section for test case (as we will lose the one that was in scope)
        auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
        SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);

        Counts assertions;
        assertions.failed = 1;
        SectionStats testCaseSectionStats(testCaseSection, assertions, 0, false);
        m_reporter->sectionEnded(testCaseSectionStats);

        auto const& testInfo = m_activeTestCase->getTestCaseInfo();

        Totals deltaTotals;
        deltaTotals.testCases.failed = 1;
        deltaTotals.assertions.failed = 1;
        m_reporter->testCaseEnded(TestCaseStats(testInfo,
                                    deltaTotals,
                                    std::string(),
                                    std::string(),
                                    false));
        m_totals.testCases.failed++;
        testGroupEnded(std::string(), m_totals, 1, 1);
        m_reporter->testRunEnded(TestRunStats(m_runInfo, m_totals, false));
    }

    bool RunContext::lastAssertionPassed() {
         return m_lastAssertionPassed;
    }

    void RunContext::assertionPassed() {
        m_lastAssertionPassed = true;
        ++m_totals.assertions.passed;
        resetAssertionInfo();
        m_messageScopes.clear();
    }

    bool RunContext::aborting() const {
        return m_totals.assertions.failed >= static_cast<std::size_t>(m_config->abortAfter());
    }

    void RunContext::runCurrentTest(std::string & redirectedCout, std::string & redirectedCerr) {
        auto const& testCaseInfo = m_activeTestCase->getTestCaseInfo();
        SectionInfo testCaseSection(testCaseInfo.lineInfo, testCaseInfo.name);
        m_reporter->sectionStarting(testCaseSection);
        Counts prevAssertions = m_totals.assertions;
        double duration = 0;
        m_shouldReportUnexpected = true;
        m_lastAssertionInfo = { "TEST_CASE"_sr, testCaseInfo.lineInfo, StringRef(),
ResultDisposition::Normal };

        seedRng(*m_config);

        Timer timer;
        CATCH_TRY {
            if (m_reporter->getPreferences().shouldRedirectStdOut) {
#if !defined(CATCH_CONFIG_EXPERIMENTAL_REDIRECT)
                RedirectedStreams redirectedStreams(redirectedCout, redirectedCerr);

                timer.start();
                invokeActiveTestCase();
#else
                OutputRedirect r(redirectedCout, redirectedCerr);
                timer.start();
                invokeActiveTestCase();
#endif
            } else {
                timer.start();
                invokeActiveTestCase();
            }
            duration = timer.getElapsedSeconds();
        } CATCH_CATCH_ANON (TestFailureException&) {
            // This just means the test was aborted due to failure
        } CATCH_CATCH_ALL {
            // Under CATCH_CONFIG_FAST_COMPILE, unexpected exceptions under REQUIRE assertions
            // are reported without translation at the point of origin.
            if( m_shouldReportUnexpected ) {
                AssertionReaction dummyReaction;
                handleUnexpectedInflightException( m_lastAssertionInfo, translateActiveException(),
dummyReaction );
            }
        }
        Counts assertions = m_totals.assertions - prevAssertions;
        bool missingAssertions = testForMissingAssertions(assertions);

        m_testCaseTracker->close();
        handleUnfinishedSections();
        m_messages.clear();
        m_messageScopes.clear();

        SectionStats testCaseSectionStats(testCaseSection, assertions, duration, missingAssertions);
        m_reporter->sectionEnded(testCaseSectionStats);
    }

    void RunContext::invokeActiveTestCase() {
        FatalConditionHandlerGuard _(&m_fatalConditionhandler);
        m_activeTestCase->invoke();
    }

    void RunContext::handleUnfinishedSections() {
        // If sections ended prematurely due to an exception we stored their
        // infos here so we can tear them down outside the unwind process.
        for (auto it = m_unfinishedSections.rbegin(),
             itEnd = m_unfinishedSections.rend();
```

```
                 it != itEnd;
                 ++it)
                sectionEnded(*it);
            m_unfinishedSections.clear();
        }

        void RunContext::handleExpr(
            AssertionInfo const& info,
            ITransientExpression const& expr,
            AssertionReaction& reaction
        ) {
            m_reporter->assertionStarting( info );

            bool negated = isFalseTest( info.resultDisposition );
            bool result = expr.getResult() != negated;

            if( result ) {
                if (!m_includeSuccessfulResults) {
                    assertionPassed();
                }
                else {
                    reportExpr(info, ResultWas::Ok, &expr, negated);
                }
            }
            else {
                reportExpr(info, ResultWas::ExpressionFailed, &expr, negated );
                populateReaction( reaction );
            }
        }
        void RunContext::reportExpr(
                AssertionInfo const &info,
                ResultWas::OfType resultType,
                ITransientExpression const *expr,
                bool negated ) {

            m_lastAssertionInfo = info;
            AssertionResultData data( resultType, LazyExpression( negated ) );

            AssertionResult assertionResult{ info, data };
            assertionResult.m_resultData.lazyExpression.m_transientExpression = expr;

            assertionEnded( assertionResult );
        }

        void RunContext::handleMessage(
                AssertionInfo const& info,
                ResultWas::OfType resultType,
                StringRef const& message,
                AssertionReaction& reaction
        ) {
            m_reporter->assertionStarting( info );

            m_lastAssertionInfo = info;

            AssertionResultData data( resultType, LazyExpression( false ) );
            data.message = static_cast<std::string>(message);
            AssertionResult assertionResult{ m_lastAssertionInfo, data };
            assertionEnded( assertionResult );
            if( !assertionResult.isOk() )
                populateReaction( reaction );
        }
        void RunContext::handleUnexpectedExceptionNotThrown(
                AssertionInfo const& info,
                AssertionReaction& reaction
        ) {
            handleNonExpr(info, Catch::ResultWas::DidntThrowException, reaction);
        }

        void RunContext::handleUnexpectedInflightException(
                AssertionInfo const& info,
                std::string const& message,
                AssertionReaction& reaction
        ) {
            m_lastAssertionInfo = info;

            AssertionResultData data( ResultWas::ThrewException, LazyExpression( false ) );
            data.message = message;
            AssertionResult assertionResult{ info, data };
            assertionEnded( assertionResult );
            populateReaction( reaction );
        }

        void RunContext::populateReaction( AssertionReaction& reaction ) {
            reaction.shouldDebugBreak = m_config->shouldDebugBreak();
            reaction.shouldThrow = aborting() || (m_lastAssertionInfo.resultDisposition &
  ResultDisposition::Normal);
        }
```

```

    void RunContext::handleIncomplete(
            AssertionInfo const& info
    ) {
        m_lastAssertionInfo = info;

        AssertionResultData data( ResultWas::ThrewException, LazyExpression( false ) );
        data.message = "Exception translation was disabled by CATCH_CONFIG_FAST_COMPILE";
        AssertionResult assertionResult{ info, data };
        assertionEnded( assertionResult );
    }
    void RunContext::handleNonExpr(
            AssertionInfo const &info,
            ResultWas::OfType resultType,
            AssertionReaction &reaction
    ) {
        m_lastAssertionInfo = info;

        AssertionResultData data( resultType, LazyExpression( false ) );
        AssertionResult assertionResult{ info, data };
        assertionEnded( assertionResult );

        if( !assertionResult.isOk() )
            populateReaction( reaction );
    }

    IResultCapture& getResultCapture() {
        if (auto* capture = getCurrentContext().getResultCapture())
            return *capture;
        else
            CATCH_INTERNAL_ERROR("No result capture instance");
    }

    void seedRng(IConfig const& config) {
        if (config.rngSeed() != 0) {
            std::srand(config.rngSeed());
            rng().seed(config.rngSeed());
        }
    }

    unsigned int rngSeed() {
        return getCurrentContext().getConfig()->rngSeed();
    }

}
// end catch_run_context.cpp
// start catch_section.cpp

namespace Catch {

    Section::Section( SectionInfo const& info )
    :   m_info( info ),
        m_sectionIncluded( getResultCapture().sectionStarted( m_info, m_assertions ) )
    {
        m_timer.start();
    }

    Section::~Section() {
        if( m_sectionIncluded ) {
            SectionEndInfo endInfo{ m_info, m_assertions, m_timer.getElapsedSeconds() };
            if( uncaught_exceptions() )
                getResultCapture().sectionEndedEarly( endInfo );
            else
                getResultCapture().sectionEnded( endInfo );
        }
    }

    // This indicates whether the section should be executed or not
    Section::operator bool() const {
        return m_sectionIncluded;
    }

} // end namespace Catch
// end catch_section.cpp
// start catch_section_info.cpp

namespace Catch {

    SectionInfo::SectionInfo
        (   SourceLineInfo const& _lineInfo,
            std::string const& _name )
    :   name( _name ),
        lineInfo( _lineInfo )
    {}

} // end namespace Catch
// end catch_section_info.cpp
// start catch_session.cpp

// start catch_session.h

#include <memory>

namespace Catch {

    class Session : NonCopyable {
    public:

        Session();
        ~Session() override;

        void showHelp() const;
        void libIdentify();

        int applyCommandLine( int argc, char const * const * argv );
    #if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
        int applyCommandLine( int argc, wchar_t const * const * argv );
    #endif

        void useConfigData( ConfigData const& configData );

        template<typename CharT>
        int run(int argc, CharT const * const argv[]) {
            if (m_startupExceptions)
                return 1;
            int returnCode = applyCommandLine(argc, argv);
            if (returnCode == 0)
                returnCode = run();
            return returnCode;
        }

        int run();

        clara::Parser const& cli() const;
        void cli( clara::Parser const& newParser );
        ConfigData& configData();
        Config& config();
    private:
        int runInternal();

        clara::Parser m_cli;
        ConfigData m_configData;
        std::shared_ptr<Config> m_config;
        bool m_startupExceptions = false;
    };

} // end namespace Catch

// end catch_session.h
// start catch_version.h

#include <iosfwd>

namespace Catch {

    // Versioning information
    struct Version {
        Version( Version const& ) = delete;
        Version& operator=( Version const& ) = delete;
        Version(    unsigned int _majorVersion,
                    unsigned int _minorVersion,
                    unsigned int _patchNumber,
                    char const * const _branchName,
                    unsigned int _buildNumber );

        unsigned int const majorVersion;
        unsigned int const minorVersion;
        unsigned int const patchNumber;

        // buildNumber is only used if branchName is not null
        char const * const branchName;
        unsigned int const buildNumber;

        friend std::ostream& operator « ( std::ostream& os, Version const& version );
    };

    Version const& libraryVersion();
}

// end catch_version.h
#include <cstdlib>
#include <iomanip>
#include <set>
#include <iterator>
```

```

namespace Catch {

    namespace {
        const int MaxExitCode = 255;

        IStreamingReporterPtr createReporter(std::string const& reporterName, IConfigPtr const&
config) {
            auto reporter = Catch::getRegistryHub().getReporterRegistry().create(reporterName,
config);
            CATCH_ENFORCE(reporter, "No reporter registered with name: '" « reporterName « "'");

            return reporter;
        }

        IStreamingReporterPtr makeReporter(std::shared_ptr<Config> const& config) {
            if (Catch::getRegistryHub().getReporterRegistry().getListeners().empty()) {
                return createReporter(config->getReporterName(), config);
            }

            // On older platforms, returning std::unique_ptr<ListeningReporter>
            // when the return type is std::unique_ptr<IStreamingReporter>
            // doesn't compile without a std::move call. However, this causes
            // a warning on newer platforms. Thus, we have to work around
            // it a bit and downcast the pointer manually.
            auto ret = std::unique_ptr<IStreamingReporter>(new ListeningReporter);
            auto& multi = static_cast<ListeningReporter&>(*ret);
            auto const& listeners = Catch::getRegistryHub().getReporterRegistry().getListeners();
            for (auto const& listener : listeners) {
                multi.addListener(listener->create(Catch::ReporterConfig(config)));
            }
            multi.addReporter(createReporter(config->getReporterName(), config));
            return ret;
        }

        class TestGroup {
        public:
            explicit TestGroup(std::shared_ptr<Config> const& config)
            : m_config{config}
            , m_context{config, makeReporter(config)}
            {
                auto const& allTestCases = getAllTestCasesSorted(*m_config);
                m_matches = m_config->testSpec().matchesByFilter(allTestCases, *m_config);
                auto const& invalidArgs = m_config->testSpec().getInvalidArgs();

                if (m_matches.empty() && invalidArgs.empty()) {
                    for (auto const& test : allTestCases)
                        if (!test.isHidden())
                            m_tests.emplace(&test);
                } else {
                    for (auto const& match : m_matches)
                        m_tests.insert(match.tests.begin(), match.tests.end());
                }
            }

            Totals execute() {
                auto const& invalidArgs = m_config->testSpec().getInvalidArgs();
                Totals totals;
                m_context.testGroupStarting(m_config->name(), 1, 1);
                for (auto const& testCase : m_tests) {
                    if (!m_context.aborting())
                        totals += m_context.runTest(*testCase);
                    else
                        m_context.reporter().skipTest(*testCase);
                }

                for (auto const& match : m_matches) {
                    if (match.tests.empty()) {
                        m_context.reporter().noMatchingTestCases(match.name);
                        totals.error = -1;
                    }
                }

                if (!invalidArgs.empty()) {
                    for (auto const& invalidArg: invalidArgs)
                         m_context.reporter().reportInvalidArguments(invalidArg);
                }

                m_context.testGroupEnded(m_config->name(), totals, 1, 1);
                return totals;
            }

        private:
            using Tests = std::set<TestCase const*>;

            std::shared_ptr<Config> m_config;
            RunContext m_context;
            Tests m_tests;
            TestSpec::Matches m_matches;
        };

        void applyFilenamesAsTags(Catch::IConfig const& config) {
            auto& tests = const_cast<std::vector<TestCase>&>(getAllTestCasesSorted(config));
            for (auto& testCase : tests) {
                auto tags = testCase.tags;

                std::string filename = testCase.lineInfo.file;
                auto lastSlash = filename.find_last_of("\\/");
                if (lastSlash != std::string::npos) {
                    filename.erase(0, lastSlash);
                    filename[0] = '#';
                }
                else
                {
                    filename.insert(0, "#");
                }

                auto lastDot = filename.find_last_of('.');
                if (lastDot != std::string::npos) {
                    filename.erase(lastDot);
                }

                tags.push_back(std::move(filename));
                setTags(testCase, tags);
            }
        }

    } // anon namespace

    Session::Session() {
        static bool alreadyInstantiated = false;
        if( alreadyInstantiated ) {
            CATCH_TRY { CATCH_INTERNAL_ERROR( "Only one instance of Catch::Session can ever be used"
); }
            CATCH_CATCH_ALL { getMutableRegistryHub().registerStartupException(); }
        }

        // There cannot be exceptions at startup in no-exception mode.
#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
        const auto& exceptions = getRegistryHub().getStartupExceptionRegistry().getExceptions();
        if ( !exceptions.empty() ) {
            config();
            getCurrentMutableContext().setConfig(m_config);

            m_startupExceptions = true;
            Colour colourGuard( Colour::Red );
            Catch::cerr() « "Errors occurred during startup!" « '\n';
            // iterate over all exceptions and notify user
            for ( const auto& ex_ptr : exceptions ) {
                try {
                    std::rethrow_exception(ex_ptr);
                } catch ( std::exception const& ex ) {
                    Catch::cerr() « Column( ex.what() ).indent(2) « '\n';
                }
            }
        }
#endif

        alreadyInstantiated = true;
        m_cli = makeCommandLineParser( m_configData );
    }
    Session::~Session() {
        Catch::cleanUp();
    }

    void Session::showHelp() const {
        Catch::cout()
                « "\nCatch v" « libraryVersion() « "\n"
                « m_cli « std::endl
                « "For more detailed usage please see the project docs\n" « std::endl;
    }
    void Session::libIdentify() {
        Catch::cout()
                « std::left « std::setw(16) « "description: " « "A Catch2 test executable\n"
                « std::left « std::setw(16) « "category: " « "testframework\n"
                « std::left « std::setw(16) « "framework: " « "Catch Test\n"
                « std::left « std::setw(16) « "version: " « libraryVersion() « std::endl;
    }

    int Session::applyCommandLine( int argc, char const * const * argv ) {
        if( m_startupExceptions )
            return 1;

        auto result = m_cli.parse( clara::Args( argc, argv ) );
        if( !result ) {
            config();
            getCurrentMutableContext().setConfig(m_config);
            Catch::cerr()
                « Colour( Colour::Red )
                « "\nError(s) in input:\n"
                « Column( result.errorMessage() ).indent( 2 )
                « "\n\n";
            Catch::cerr() « "Run with -? for usage\n" « std::endl;
            return MaxExitCode;
        }

        if( m_configData.showHelp )
            showHelp();
        if( m_configData.libIdentify )
            libIdentify();
        m_config.reset();
        return 0;
    }

#if defined(CATCH_CONFIG_WCHAR) && defined(_WIN32) && defined(UNICODE)
    int Session::applyCommandLine( int argc, wchar_t const * const * argv ) {

        char **utf8Argv = new char *[ argc ];

        for ( int i = 0; i < argc; ++i ) {
            int bufSize = WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, nullptr, 0, nullptr, nullptr
);

            utf8Argv[ i ] = new char[ bufSize ];

            WideCharToMultiByte( CP_UTF8, 0, argv[i], -1, utf8Argv[i], bufSize, nullptr, nullptr );
        }

        int returnCode = applyCommandLine( argc, utf8Argv );

        for ( int i = 0; i < argc; ++i )
            delete [] utf8Argv[ i ];

        delete [] utf8Argv;

        return returnCode;
    }
#endif

    void Session::useConfigData( ConfigData const& configData ) {
        m_configData = configData;
        m_config.reset();
    }

    int Session::run() {
        if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeStart ) != 0 ) {
            Catch::cout() « "...waiting for enter/ return before starting" « std::endl;
            static_cast<void>(std::getchar());
        }
        int exitCode = runInternal();
        if( ( m_configData.waitForKeypress & WaitForKeypress::BeforeExit ) != 0 ) {
            Catch::cout() « "...waiting for enter/ return before exiting, with code: " « exitCode «
std::endl;
            static_cast<void>(std::getchar());
        }
        return exitCode;
    }

    clara::Parser const& Session::cli() const {
        return m_cli;
    }
    void Session::cli( clara::Parser const& newParser ) {
        m_cli = newParser;
    }
    ConfigData& Session::configData() {
        return m_configData;
    }
    Config& Session::config() {
        if( !m_config )
            m_config = std::make_shared<Config>( m_configData );
        return *m_config;
    }

    int Session::runInternal() {
        if( m_startupExceptions )
            return 1;

        if (m_configData.showHelp || m_configData.libIdentify) {
            return 0;
        }

        CATCH_TRY {
            config(); // Force config to be constructed

            seedRng( *m_config );

            if( m_configData.filenamesAsTags )
                applyFilenamesAsTags( *m_config );

            // Handle list request
            if( Option<std::size_t> listed = list( m_config ) )
                return (std::min) (MaxExitCode, static_cast<int>(*listed));

            TestGroup tests { m_config };
            auto const totals = tests.execute();

            if( m_config->warnAboutNoTests() && totals.error == -1 )
                return 2;

            // Note that on unices only the lower 8 bits are usually used, clamping
            // the return value to 255 prevents false negative when some multiple
            // of 256 tests has failed
            return (std::min) (MaxExitCode, (std::max) (totals.error,
static_cast<int>(totals.assertions.failed)));
        }
#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
        catch( std::exception& ex ) {
            Catch::cerr() « ex.what() « std::endl;
            return MaxExitCode;
        }
#endif
    }

} // end namespace Catch
// end catch_session.cpp
// start catch_singletons.cpp

#include <vector>

namespace Catch {

    namespace {
        static auto getSingletons() -> std::vector<ISingleton*>*& {
            static std::vector<ISingleton*>* g_singletons = nullptr;
            if( !g_singletons )
                g_singletons = new std::vector<ISingleton*>();
            return g_singletons;
        }
    }

    ISingleton::~ISingleton() {}

    void addSingleton(ISingleton* singleton ) {
        getSingletons()->push_back( singleton );
    }
    void cleanupSingletons() {
        auto& singletons = getSingletons();
        for( auto singleton : *singletons )
            delete singleton;
        delete singletons;
        singletons = nullptr;
    }

} // namespace Catch
// end catch_singletons.cpp
// start catch_startup_exception_registry.cpp

#if !defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
namespace Catch {
void StartupExceptionRegistry::add( std::exception_ptr const& exception ) noexcept {
        CATCH_TRY {
            m_exceptions.push_back(exception);
        } CATCH_CATCH_ALL {
            // If we run out of memory during start-up there's really not a lot more we can do about
it
            std::terminate();
        }
    }

    std::vector<std::exception_ptr> const& StartupExceptionRegistry::getExceptions() const noexcept {
        return m_exceptions;
    }

} // end namespace Catch
#endif
// end catch_startup_exception_registry.cpp
// start catch_stream.cpp

```

```
#include <cstdio>
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <memory>

namespace Catch {

    Catch::IStream::~IStream() = default;

    namespace Detail { namespace {
        template<typename WriterF, std::size_t bufferSize=256>
        class StreamBufImpl : public std::streambuf {
            char data[bufferSize];
            WriterF m_writer;

        public:
            StreamBufImpl() {
                setp( data, data + sizeof(data) );
            }

            ~StreamBufImpl() noexcept {
                StreamBufImpl::sync();
            }

        private:
            int overflow( int c ) override {
                sync();

                if( c != EOF ) {
                    if( pbase() == epptr() )
                        m_writer( std::string( 1, static_cast<char>( c ) ) );
                    else
                        sputc( static_cast<char>( c ) );
                }
                return 0;
            }

            int sync() override {
                if( pbase() != pptr() ) {
                    m_writer( std::string( pbase(), static_cast<std::string::size_type>( pptr() -
pbase() ) ) );
                    setp( pbase(), epptr() );
                }
                return 0;
            }
        };


        struct OutputDebugWriter {

            void operator()( std::string const&str ) {
                writeToDebugConsole( str );
            }
        };


        class FileStream : public IStream {
            mutable std::ofstream m_ofs;
        public:
            FileStream( StringRef filename ) {
                m_ofs.open( filename.c_str() );
                CATCH_ENFORCE( !m_ofs.fail(), "Unable to open file: '" « filename « "'" );
            }
            ~FileStream() override = default;
        public: // IStream
            std::ostream& stream() const override {
                return m_ofs;
            }
        };


        class CoutStream : public IStream {
            mutable std::ostream m_os;
        public:
            // Store the streambuf from cout up-front because
            // cout may get redirected when running tests
            CoutStream() : m_os( Catch::cout().rdbuf() ) {}
            ~CoutStream() override = default;

        public: // IStream
            std::ostream& stream() const override { return m_os; }
        };


        class DebugOutStream : public IStream {
            std::unique_ptr<StreamBufImpl<OutputDebugWriter» m_streamBuf;
            mutable std::ostream m_os;
        public:
            DebugOutStream()
            :   m_streamBuf( new StreamBufImpl<OutputDebugWriter>() ),
                m_os( m_streamBuf.get() )
            {}

            ~DebugOutStream() override = default;

        public: // IStream
            std::ostream& stream() const override { return m_os; }
        };

    }} // namespace anon::detail


    auto makeStream( StringRef const &filename ) -> IStream const* {
        if( filename.empty() )
            return new Detail::CoutStream();
        else if( filename[0] == '%' ) {
            if( filename == "%debug" )
                return new Detail::DebugOutStream();
            else
                CATCH_ERROR( "Unrecognised stream: '" « filename « "'" );
        }
        else
            return new Detail::FileStream( filename );
    }

    // This class encapsulates the idea of a pool of ostringstreams that can be reused.
    struct StringStreams {
        std::vector<std::unique_ptr<std::ostringstream» m_streams;
        std::vector<std::size_t> m_unused;
        std::ostringstream m_referenceStream; // Used for copy state/ flags from

        auto add() -> std::size_t {
            if( m_unused.empty() ) {
                m_streams.push_back( std::unique_ptr<std::ostringstream>( new std::ostringstream ) );
                return m_streams.size()-1;
            }
            else {
                auto index = m_unused.back();
                m_unused.pop_back();
                return index;
            }
        }

        void release( std::size_t index ) {
            m_streams[index]->copyfmt( m_referenceStream ); // Restore initial flags and other state
            m_unused.push_back(index);
        }
    };

    ReusableStringStream::ReusableStringStream()
    :   m_index( Singleton<StringStreams>::getMutable().add() ),
        m_oss( Singleton<StringStreams>::getMutable().m_streams[m_index].get() )
    {}

    ReusableStringStream::~ReusableStringStream() {
        static_cast<std::ostringstream*>( m_oss )->str("");
        m_oss->clear();
        Singleton<StringStreams>::getMutable().release( m_index );
    }

    auto ReusableStringStream::str() const -> std::string {
        return static_cast<std::ostringstream*>( m_oss )->str();
    }


#ifndef CATCH_CONFIG_NOSTDOUT // If you #define this you must implement these functions
    std::ostream& cout() { return std::cout; }
    std::ostream& cerr() { return std::cerr; }
    std::ostream& clog() { return std::clog; }
#endif
}
// end catch_stream.cpp
// start catch_string_manip.cpp

#include <algorithm>
#include <ostream>
#include <cstring>
#include <cctype>
#include <vector>

namespace Catch {

```

```
namespace {
    char toLowerCh(char c) {
        return static_cast<char>( std::tolower( static_cast<unsigned char>(c) ) );
    }
}

bool startsWith( std::string const& s, std::string const& prefix ) {
    return s.size() >= prefix.size() && std::equal(prefix.begin(), prefix.end(), s.begin());
}
bool startsWith( std::string const& s, char prefix ) {
    return !s.empty() && s[0] == prefix;
}
bool endsWith( std::string const& s, std::string const& suffix ) {
    return s.size() >= suffix.size() && std::equal(suffix.rbegin(), suffix.rend(), s.rbegin());
}
bool endsWith( std::string const& s, char suffix ) {
    return !s.empty() && s[s.size()-1] == suffix;
}
bool contains( std::string const& s, std::string const& infix ) {
    return s.find( infix ) != std::string::npos;
}
void toLowerInPlace( std::string& s ) {
    std::transform( s.begin(), s.end(), s.begin(), toLowerCh );
}
std::string toLower( std::string const& s ) {
    std::string lc = s;
    toLowerInPlace( lc );
    return lc;
}
std::string trim( std::string const& str ) {
    static char const* whitespaceChars = "\n\r\t ";
    std::string::size_type start = str.find_first_not_of( whitespaceChars );
    std::string::size_type end = str.find_last_not_of( whitespaceChars );

    return start != std::string::npos ? str.substr( start, 1+end-start ) : std::string();
}

StringRef trim(StringRef ref) {
    const auto is_ws = [](char c) {
        return c == ' ' || c == '\t' || c == '\n' || c == '\r';
    };
    size_t real_begin = 0;
    while (real_begin < ref.size() && is_ws(ref[real_begin])) { ++real_begin; }
    size_t real_end = ref.size();
    while (real_end > real_begin && is_ws(ref[real_end - 1])) { --real_end; }

    return ref.substr(real_begin, real_end - real_begin);
}

bool replaceInPlace( std::string& str, std::string const& replaceThis, std::string const& withThis
) {
    bool replaced = false;
    std::size_t i = str.find( replaceThis );
    while( i != std::string::npos ) {
        replaced = true;
        str = str.substr( 0, i ) + withThis + str.substr( i+replaceThis.size() );
        if( i < str.size()-withThis.size() )
            i = str.find( replaceThis, i+withThis.size() );
        else
            i = std::string::npos;
    }
    return replaced;
}

std::vector<StringRef> splitStringRef( StringRef str, char delimiter ) {
    std::vector<StringRef> subStrings;
    std::size_t start = 0;
    for(std::size_t pos = 0; pos < str.size(); ++pos ) {
        if( str[pos] == delimiter ) {
            if( pos - start > 1 )
                subStrings.push_back( str.substr( start, pos-start ) );
            start = pos+1;
        }
    }
    if( start < str.size() )
        subStrings.push_back( str.substr( start, str.size()-start ) );
    return subStrings;
}

pluralise::pluralise( std::size_t count, std::string const& label )
:   m_count( count ),
    m_label( label )
{}

std::ostream& operator « ( std::ostream& os, pluralise const& pluraliser ) {
    os « pluraliser.m_count « ' ' « pluraliser.m_label;
    if( pluraliser.m_count != 1 )
```

```
            os « 's';
        return os;
    }

}
// end catch_string_manip.cpp
// start catch_stringref.cpp

#include <algorithm>
#include <ostream>
#include <cstring>
#include <cstdint>

namespace Catch {
    StringRef::StringRef( char const* rawChars ) noexcept
    : StringRef( rawChars, static_cast<StringRef::size_type>(std::strlen(rawChars) ) )
    {}

    auto StringRef::c_str() const -> char const* {
        CATCH_ENFORCE(isNullTerminated(), "Called StringRef::c_str() on a non-null-terminated
instance");
        return m_start;
    }
    auto StringRef::data() const noexcept -> char const* {
        return m_start;
    }

    auto StringRef::substr( size_type start, size_type size ) const noexcept -> StringRef {
        if (start < m_size) {
            return StringRef(m_start + start, (std::min)(m_size - start, size));
        } else {
            return StringRef();
        }
    }
    auto StringRef::operator == ( StringRef const& other ) const noexcept -> bool {
        return m_size == other.m_size
            && (std::memcmp( m_start, other.m_start, m_size ) == 0);
    }

    auto operator « ( std::ostream& os, StringRef const& str ) -> std::ostream& {
        return os.write(str.data(), str.size());
    }

    auto operator+=( std::string& lhs, StringRef const& rhs ) -> std::string& {
        lhs.append(rhs.data(), rhs.size());
        return lhs;
    }

} // namespace Catch
// end catch_stringref.cpp
// start catch_tag_alias.cpp

namespace Catch {
    TagAlias::TagAlias(std::string const & _tag, SourceLineInfo _lineInfo): tag(_tag),
lineInfo(_lineInfo) {}
}
// end catch_tag_alias.cpp
// start catch_tag_alias_autoregistrar.cpp

namespace Catch {

    RegistrarForTagAliases::RegistrarForTagAliases(char const* alias, char const* tag, SourceLineInfo
const& lineInfo) {
        CATCH_TRY {
            getMutableRegistryHub().registerTagAlias(alias, tag, lineInfo);
        } CATCH_CATCH_ALL {
            // Do not throw when constructing global objects, instead register the exception to be
processed later
            getMutableRegistryHub().registerStartupException();
        }
    }

}
// end catch_tag_alias_autoregistrar.cpp
// start catch_tag_alias_registry.cpp

#include <sstream>

namespace Catch {

    TagAliasRegistry::~TagAliasRegistry() {}

    TagAlias const* TagAliasRegistry::find( std::string const& alias ) const {
        auto it = m_registry.find( alias );
        if( it != m_registry.end() )
            return &(it->second);
        else
```

```
            return nullptr;
    }

    std::string TagAliasRegistry::expandAliases( std::string const& unexpandedTestSpec ) const {
        std::string expandedTestSpec = unexpandedTestSpec;
        for( auto const& registryKvp : m_registry ) {
            std::size_t pos = expandedTestSpec.find( registryKvp.first );
            if( pos != std::string::npos ) {
                expandedTestSpec =  expandedTestSpec.substr( 0, pos ) +
                                    registryKvp.second.tag +
                                    expandedTestSpec.substr( pos + registryKvp.first.size() );
            }
        }
        return expandedTestSpec;
    }

    void TagAliasRegistry::add( std::string const& alias, std::string const& tag, SourceLineInfo
const& lineInfo ) {
        CATCH_ENFORCE( startsWith(alias, "[@") && endsWith(alias, ']'),
                      "error: tag alias, '" « alias « "' is not of the form [@alias name].\n" «
lineInfo );

        CATCH_ENFORCE( m_registry.insert(std::make_pair(alias, TagAlias(tag, lineInfo))).second,
                      "error: tag alias, '" « alias « "' already registered.\n"
                      « "\tFirst seen at: " « find(alias)->lineInfo « "\n"
                      « "\tRedefined at: " « lineInfo );
    }

    ITagAliasRegistry::~ITagAliasRegistry() {}

    ITagAliasRegistry const& ITagAliasRegistry::get() {
        return getRegistryHub().getTagAliasRegistry();
    }

} // end namespace Catch
// end catch_tag_alias_registry.cpp
// start catch_test_case_info.cpp

#include <cctype>
#include <exception>
#include <algorithm>
#include <sstream>

namespace Catch {

    namespace {
        TestCaseInfo::SpecialProperties parseSpecialTag( std::string const& tag ) {
            if( startsWith( tag, '.' ) ||
                tag == "!hide" )
                return TestCaseInfo::IsHidden;
            else if( tag == "!throws" )
                return TestCaseInfo::Throws;
            else if( tag == "!shouldfail" )
                return TestCaseInfo::ShouldFail;
            else if( tag == "!mayfail" )
                return TestCaseInfo::MayFail;
            else if( tag == "!nonportable" )
                return TestCaseInfo::NonPortable;
            else if( tag == "!benchmark" )
                return static_cast<TestCaseInfo::SpecialProperties>( TestCaseInfo::Benchmark |
TestCaseInfo::IsHidden );
            else
                return TestCaseInfo::None;
        }
        bool isReservedTag( std::string const& tag ) {
            return parseSpecialTag( tag ) == TestCaseInfo::None && tag.size() > 0 && !std::isalnum(
static_cast<unsigned char>(tag[0]) );
        }
        void enforceNotReservedTag( std::string const& tag, SourceLineInfo const& _lineInfo ) {
            CATCH_ENFORCE( !isReservedTag(tag),
                          "Tag name: [" « tag « "] is not allowed.\n"
                          « "Tag names starting with non alphanumeric characters are reserved\n"
                          « _lineInfo );
        }
    }

    TestCase makeTestCase(  ITestInvoker* _testCase,
                            std::string const& _className,
                            NameAndTags const& nameAndTags,
                            SourceLineInfo const& _lineInfo )
    {
        bool isHidden = false;

        // Parse out tags
        std::vector<std::string> tags;
        std::string desc, tag;
        bool inTag = false;
        for (char c : nameAndTags.tags) {
            if( !inTag ) {
                if( c == '[' )
                    inTag = true;
                else
                    desc += c;
            }
            else {
                if( c == ']' ) {
                    TestCaseInfo::SpecialProperties prop = parseSpecialTag( tag );
                    if( ( prop & TestCaseInfo::IsHidden ) != 0 )
                        isHidden = true;
                    else if( prop == TestCaseInfo::None )
                        enforceNotReservedTag( tag, _lineInfo );

                    // Merged hide tags like `[.approvals]` should be added as
                    // `[.][approvals]`. The `[.]` is added at later point, so
                    // we only strip the prefix
                    if (startsWith(tag, '.') && tag.size() > 1) {
                        tag.erase(0, 1);
                    }
                    tags.push_back( tag );
                    tag.clear();
                    inTag = false;
                }
                else
                    tag += c;
            }
        }
        if( isHidden ) {
            // Add all "hidden" tags to make them behave identically
            tags.insert( tags.end(), { ".", "!hide" } );
        }

        TestCaseInfo info( static_cast<std::string>(nameAndTags.name), _className, desc, tags,
_lineInfo );
        return TestCase( _testCase, std::move(info) );
    }

    void setTags( TestCaseInfo& testCaseInfo, std::vector<std::string> tags ) {
        std::sort(begin(tags), end(tags));
        tags.erase(std::unique(begin(tags), end(tags)), end(tags));
        testCaseInfo.lcaseTags.clear();

        for( auto const& tag : tags ) {
            std::string lcaseTag = toLower( tag );
            testCaseInfo.properties = static_cast<TestCaseInfo::SpecialProperties>(
testCaseInfo.properties | parseSpecialTag( lcaseTag ) );
            testCaseInfo.lcaseTags.push_back( lcaseTag );
        }
        testCaseInfo.tags = std::move(tags);
    }

    TestCaseInfo::TestCaseInfo( std::string const& _name,
                                std::string const& _className,
                                std::string const& _description,
                                std::vector<std::string> const& _tags,
                                SourceLineInfo const& _lineInfo )
    :   name( _name ),
        className( _className ),
        description( _description ),
        lineInfo( _lineInfo ),
        properties( None )
    {
        setTags( *this, _tags );
    }

    bool TestCaseInfo::isHidden() const {
        return ( properties & IsHidden ) != 0;
    }
    bool TestCaseInfo::throws() const {
        return ( properties & Throws ) != 0;
    }
    bool TestCaseInfo::okToFail() const {
        return ( properties & (ShouldFail | MayFail ) ) != 0;
    }
    bool TestCaseInfo::expectedToFail() const {
        return ( properties & (ShouldFail ) ) != 0;
    }

    std::string TestCaseInfo::tagsAsString() const {
        std::string ret;
        // '[' and ']' per tag
        std::size_t full_size = 2 * tags.size();
        for (const auto& tag : tags) {
            full_size += tag.size();
        }
        ret.reserve(full_size);
        for (const auto& tag : tags) {
            ret.push_back('[');
            ret.append(tag);
            ret.push_back(']');
        }

        return ret;
    }

    TestCase::TestCase( ITestInvoker* testCase, TestCaseInfo&& info ) : TestCaseInfo( std::move(info)
), test( testCase ) {}

    TestCase TestCase::withName( std::string const& _newName ) const {
        TestCase other( *this );
        other.name = _newName;
        return other;
    }

    void TestCase::invoke() const {
        test->invoke();
    }

    bool TestCase::operator == ( TestCase const& other ) const {
        return  test.get() == other.test.get() &&
                name == other.name &&
                className == other.className;
    }

    bool TestCase::operator < ( TestCase const& other ) const {
        return name < other.name;
    }

    TestCaseInfo const& TestCase::getTestCaseInfo() const
    {
        return *this;
    }

} // end namespace Catch
// end catch_test_case_info.cpp
// start catch_test_case_registry_impl.cpp

#include <algorithm>
#include <sstream>

namespace Catch {

    namespace {
        struct TestHasher {
            using hash_t = uint64_t;

            explicit TestHasher( hash_t hashSuffix ):
                m_hashSuffix{ hashSuffix } {}

            uint32_t operator()( TestCase const& t ) const {
                // FNV-1a hash with multiplication fold.
                const hash_t prime = 1099511628211u;
                hash_t hash = 14695981039346656037u;
                for ( const char c : t.name ) {
                    hash ^= c;
                    hash *= prime;
                }
                hash ^= m_hashSuffix;
                hash *= prime;
                const uint32_t low{ static_cast<uint32_t>( hash ) };
                const uint32_t high{ static_cast<uint32_t>( hash » 32 ) };
                return low * high;
            }

        private:
            hash_t m_hashSuffix;
        };
    } // end unnamed namespace

    std::vector<TestCase> sortTests( IConfig const& config, std::vector<TestCase> const&
unsortedTestCases ) {
        switch( config.runOrder() ) {
            case RunTests::InDeclarationOrder:
                // already in declaration order
                break;

            case RunTests::InLexicographicalOrder: {
                std::vector<TestCase> sorted = unsortedTestCases;
                std::sort( sorted.begin(), sorted.end() );
                return sorted;
            }

            case RunTests::InRandomOrder: {
                seedRng( config );
                TestHasher h{ config.rngSeed() };

                using hashedTest = std::pair<TestHasher::hash_t, TestCase const*>;
                std::vector<hashedTest> indexed_tests;
                indexed_tests.reserve( unsortedTestCases.size() );

                for (auto const& testCase : unsortedTestCases) {
                    indexed_tests.emplace_back(h(testCase), &testCase);
                }

                std::sort(indexed_tests.begin(), indexed_tests.end(),
                          [](hashedTest const& lhs, hashedTest const& rhs) {
                          if (lhs.first == rhs.first) {
                              return lhs.second->name < rhs.second->name;
                          }
                          return lhs.first < rhs.first;
                });

                std::vector<TestCase> sorted;
                sorted.reserve( indexed_tests.size() );

                for (auto const& hashed : indexed_tests) {
                    sorted.emplace_back(*hashed.second);
                }

                return sorted;
            }
        }
        return unsortedTestCases;
    }

    bool isThrowSafe( TestCase const& testCase, IConfig const& config ) {
        return !testCase.throws() || config.allowThrows();
    }

    bool matchTest( TestCase const& testCase, TestSpec const& testSpec, IConfig const& config ) {
        return testSpec.matches( testCase ) && isThrowSafe( testCase, config );
    }

    void enforceNoDuplicateTestCases( std::vector<TestCase> const& functions ) {
        std::set<TestCase> seenFunctions;
        for( auto const& function : functions ) {
            auto prev = seenFunctions.insert( function );
            CATCH_ENFORCE( prev.second,
                    "error: TEST_CASE( \"" « function.name « "\" ) already defined.\n"
                    « "\tFirst seen at " « prev.first->getTestCaseInfo().lineInfo « "\n"
                    « "\tRedefined at " « function.getTestCaseInfo().lineInfo );
        }
    }

    std::vector<TestCase> filterTests( std::vector<TestCase> const& testCases, TestSpec const&
testSpec, IConfig const& config ) {
        std::vector<TestCase> filtered;
        filtered.reserve( testCases.size() );
        for (auto const& testCase : testCases) {
            if ((!testSpec.hasFilters() && !testCase.isHidden()) ||
                (testSpec.hasFilters() && matchTest(testCase, testSpec, config))) {
                filtered.push_back(testCase);
            }
        }
        return filtered;
    }
    std::vector<TestCase> const& getAllTestCasesSorted( IConfig const& config ) {
        return getRegistryHub().getTestCaseRegistry().getAllTestsSorted( config );
    }

    void TestRegistry::registerTest( TestCase const& testCase ) {
        std::string name = testCase.getTestCaseInfo().name;
        if( name.empty() ) {
            ReusableStringStream rss;
            rss « "Anonymous test case " « ++m_unnamedCount;
            return registerTest( testCase.withName( rss.str() ) );
        }
        m_functions.push_back( testCase );
    }

    std::vector<TestCase> const& TestRegistry::getAllTests() const {
        return m_functions;
    }
    std::vector<TestCase> const& TestRegistry::getAllTestsSorted( IConfig const& config ) const {
        if( m_sortedFunctions.empty() )
            enforceNoDuplicateTestCases( m_functions );

        if(  m_currentSortOrder != config.runOrder() || m_sortedFunctions.empty() ) {
            m_sortedFunctions = sortTests( config, m_functions );
            m_currentSortOrder = config.runOrder();
        }
        return m_sortedFunctions;
    }

    TestInvokerAsFunction::TestInvokerAsFunction( void(*testAsFunction)() ) noexcept :
m_testAsFunction( testAsFunction ) {}

    void TestInvokerAsFunction::invoke() const {
        m_testAsFunction();
    }

    std::string extractClassName( StringRef const& classOrQualifiedMethodName ) {
        std::string className(classOrQualifiedMethodName);
        if( startsWith( className, '&' ) )
        {
            std::size_t lastColons = className.rfind( "::" );
            std::size_t penultimateColons = className.rfind( "::", lastColons-1 );
            if( penultimateColons == std::string::npos )
                penultimateColons = 1;
            className = className.substr( penultimateColons, lastColons-penultimateColons );
        }
        return className;
    }

} // end namespace Catch
// end catch_test_case_registry_impl.cpp
// start catch_test_case_tracker.cpp

#include <algorithm>
#include <cassert>
#include <stdexcept>
#include <memory>
#include <sstream>

#if defined(__clang__)
#    pragma clang diagnostic push
#    pragma clang diagnostic ignored "-Wexit-time-destructors"
#endif

namespace Catch {
namespace TestCaseTracking {

    NameAndLocation::NameAndLocation( std::string const& _name, SourceLineInfo const& _location )
    :   name( _name ),
        location( _location )
    {}

    ITracker::~ITracker() = default;

    ITracker& TrackerContext::startRun() {
        m_rootTracker = std::make_shared<SectionTracker>( NameAndLocation( "{root}",
CATCH_INTERNAL_LINEINFO ), *this, nullptr );
        m_currentTracker = nullptr;
        m_runState = Executing;
        return *m_rootTracker;
    }

    void TrackerContext::endRun() {
        m_rootTracker.reset();
        m_currentTracker = nullptr;
        m_runState = NotStarted;
    }

    void TrackerContext::startCycle() {
        m_currentTracker = m_rootTracker.get();
        m_runState = Executing;
    }
    void TrackerContext::completeCycle() {
        m_runState = CompletedCycle;
    }

    bool TrackerContext::completedCycle() const {
        return m_runState == CompletedCycle;
    }
    ITracker& TrackerContext::currentTracker() {
        return *m_currentTracker;
    }
    void TrackerContext::setCurrentTracker( ITracker* tracker ) {
        m_currentTracker = tracker;
    }

    TrackerBase::TrackerBase( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker*
parent ):
        ITracker(nameAndLocation),
        m_ctx( ctx ),
        m_parent( parent )
    {}

    bool TrackerBase::isComplete() const {
        return m_runState == CompletedSuccessfully || m_runState == Failed;
    }
    bool TrackerBase::isSuccessfullyCompleted() const {
        return m_runState == CompletedSuccessfully;
    }
    bool TrackerBase::isOpen() const {
        return m_runState != NotStarted && !isComplete();
    }
    bool TrackerBase::hasChildren() const {
        return !m_children.empty();
    }

    void TrackerBase::addChild( ITrackerPtr const& child ) {
        m_children.push_back( child );
    }

    ITrackerPtr TrackerBase::findChild( NameAndLocation const& nameAndLocation ) {
        auto it = std::find_if( m_children.begin(), m_children.end(),
            [&nameAndLocation]( ITrackerPtr const& tracker ){
                return
                    tracker->nameAndLocation().location == nameAndLocation.location &&
                    tracker->nameAndLocation().name == nameAndLocation.name;
            } );
        return( it != m_children.end() )
            ? *it
            : nullptr;
    }
    ITracker& TrackerBase::parent() {
        assert( m_parent ); // Should always be non-null except for root
        return *m_parent;
    }

    void TrackerBase::openChild() {
        if( m_runState != ExecutingChildren ) {
            m_runState = ExecutingChildren;
            if( m_parent )
                m_parent->openChild();
        }
    }

    bool TrackerBase::isSectionTracker() const { return false; }
    bool TrackerBase::isGeneratorTracker() const { return false; }

    void TrackerBase::open() {
        m_runState = Executing;
        moveToThis();
        if( m_parent )
            m_parent->openChild();
    }

    void TrackerBase::close() {

        // Close any still open children (e.g. generators)
        while( &m_ctx.currentTracker() != this )
            m_ctx.currentTracker().close();

        switch( m_runState ) {
            case NeedsAnotherRun:
                break;

            case Executing:
                m_runState = CompletedSuccessfully;
                break;
            case ExecutingChildren:
                if( std::all_of(m_children.begin(), m_children.end(), [](ITrackerPtr const& t){ return
  t->isComplete(); }) )
                    m_runState = CompletedSuccessfully;
                break;

            case NotStarted:
            case CompletedSuccessfully:
            case Failed:
                CATCH_INTERNAL_ERROR( "Illogical state: " « m_runState );

            default:
                CATCH_INTERNAL_ERROR( "Unknown state: " « m_runState );
        }
        moveToParent();
        m_ctx.completeCycle();
    }
    void TrackerBase::fail() {
        m_runState = Failed;
        if( m_parent )
            m_parent->markAsNeedingAnotherRun();
        moveToParent();
        m_ctx.completeCycle();
    }
    void TrackerBase::markAsNeedingAnotherRun() {
        m_runState = NeedsAnotherRun;
    }

    void TrackerBase::moveToParent() {
        assert( m_parent );
        m_ctx.setCurrentTracker( m_parent );
    }
    void TrackerBase::moveToThis() {
        m_ctx.setCurrentTracker( this );
    }

    SectionTracker::SectionTracker( NameAndLocation const& nameAndLocation, TrackerContext& ctx, ITracker* parent )
    :   TrackerBase( nameAndLocation, ctx, parent ),
        m_trimmed_name(trim(nameAndLocation.name))
    {
        if( parent ) {
            while( !parent->isSectionTracker() )
                parent = &parent->parent();

            SectionTracker& parentSection = static_cast<SectionTracker&>( *parent );
            addNextFilters( parentSection.m_filters );
        }
    }

    bool SectionTracker::isComplete() const {
        bool complete = true;

        if (m_filters.empty()
            || m_filters[0] == ""
            || std::find(m_filters.begin(), m_filters.end(), m_trimmed_name) != m_filters.end()) {
            complete = TrackerBase::isComplete();
        }
        return complete;
    }

    bool SectionTracker::isSectionTracker() const { return true; }

    SectionTracker& SectionTracker::acquire( TrackerContext& ctx, NameAndLocation const& nameAndLocation ) {
        std::shared_ptr<SectionTracker> section;

        ITracker& currentTracker = ctx.currentTracker();
        if( ITrackerPtr childTracker = currentTracker.findChild( nameAndLocation ) ) {
            assert( childTracker );
            assert( childTracker->isSectionTracker() );
            section = std::static_pointer_cast<SectionTracker>( childTracker );
        }
        else {
            section = std::make_shared<SectionTracker>( nameAndLocation, ctx, &currentTracker );
            currentTracker.addChild( section );
        }
        if( !ctx.completedCycle() )
            section->tryOpen();
        return *section;
    }

    void SectionTracker::tryOpen() {
        if( !isComplete() )
            open();
    }

    void SectionTracker::addInitialFilters( std::vector<std::string> const& filters ) {
        if( !filters.empty() ) {
            m_filters.reserve( m_filters.size() + filters.size() + 2 );
            m_filters.emplace_back(""); // Root - should never be consulted
            m_filters.emplace_back(""); // Test Case - not a section filter
            m_filters.insert( m_filters.end(), filters.begin(), filters.end() );
        }
    }
    void SectionTracker::addNextFilters( std::vector<std::string> const& filters ) {
        if( filters.size() > 1 )
            m_filters.insert( m_filters.end(), filters.begin()+1, filters.end() );
    }

    std::vector<std::string> const& SectionTracker::getFilters() const {
        return m_filters;
    }

    std::string const& SectionTracker::trimmedName() const {
        return m_trimmed_name;
    }

```

```
} // namespace TestCaseTracking

using TestCaseTracking::ITracker;
using TestCaseTracking::TrackerContext;
using TestCaseTracking::SectionTracker;

} // namespace Catch

#if defined(__clang__)
#    pragma clang diagnostic pop
#endif
// end catch_test_case_tracker.cpp
// start catch_test_registry.cpp

namespace Catch {

    auto makeTestInvoker( void(*testAsFunction)() ) noexcept -> ITestInvoker* {
        return new(std::nothrow) TestInvokerAsFunction( testAsFunction );
    }

    NameAndTags::NameAndTags( StringRef const& name_ , StringRef const& tags_ ) noexcept : name( name_
), tags( tags_ ) {}

    AutoReg::AutoReg( ITestInvoker* invoker, SourceLineInfo const& lineInfo, StringRef const&
classOrMethod, NameAndTags const& nameAndTags ) noexcept {
        CATCH_TRY {
            getMutableRegistryHub()
                    .registerTest(
                        makeTestCase(
                            invoker,
                            extractClassName( classOrMethod ),
                            nameAndTags,
                            lineInfo));
        } CATCH_CATCH_ALL {
            // Do not throw when constructing global objects, instead register the exception to be
processed later
            getMutableRegistryHub().registerStartupException();
        }
    }

    AutoReg::~AutoReg() = default;
}
// end catch_test_registry.cpp
// start catch_test_spec.cpp

#include <algorithm>
#include <string>
#include <vector>
#include <memory>

namespace Catch {

    TestSpec::Pattern::Pattern( std::string const& name )
    : m_name( name )
    {}

    TestSpec::Pattern::~Pattern() = default;

    std::string const& TestSpec::Pattern::name() const {
        return m_name;
    }

    TestSpec::NamePattern::NamePattern( std::string const& name, std::string const& filterString )
    : Pattern( filterString )
    , m_wildcardPattern( toLower( name ), CaseSensitive::No )
    {}

    bool TestSpec::NamePattern::matches( TestCaseInfo const& testCase ) const {
        return m_wildcardPattern.matches( testCase.name );
    }

    TestSpec::TagPattern::TagPattern( std::string const& tag, std::string const& filterString )
    : Pattern( filterString )
    , m_tag( toLower( tag ) )
    {}

    bool TestSpec::TagPattern::matches( TestCaseInfo const& testCase ) const {
        return std::find(begin(testCase.lcaseTags),
                         end(testCase.lcaseTags),
                         m_tag) != end(testCase.lcaseTags);
    }

    TestSpec::ExcludedPattern::ExcludedPattern( PatternPtr const& underlyingPattern )
    : Pattern( underlyingPattern->name() )
    , m_underlyingPattern( underlyingPattern )
    {}

    bool TestSpec::ExcludedPattern::matches( TestCaseInfo const& testCase ) const {
        return !m_underlyingPattern->matches( testCase );
    }

    bool TestSpec::Filter::matches( TestCaseInfo const& testCase ) const {
        return std::all_of( m_patterns.begin(), m_patterns.end(), [&]( PatternPtr const& p ){ return
p->matches( testCase ); } );
    }

    std::string TestSpec::Filter::name() const {
        std::string name;
        for( auto const& p : m_patterns )
            name += p->name();
        return name;
    }

    bool TestSpec::hasFilters() const {
        return !m_filters.empty();
    }

    bool TestSpec::matches( TestCaseInfo const& testCase ) const {
        return std::any_of( m_filters.begin(), m_filters.end(), [&]( Filter const& f ){ return
f.matches( testCase ); } );
    }

    TestSpec::Matches TestSpec::matchesByFilter( std::vector<TestCase> const& testCases, IConfig
const& config ) const
    {
        Matches matches( m_filters.size() );
        std::transform( m_filters.begin(), m_filters.end(), matches.begin(), [&]( Filter const& filter
){
            std::vector<TestCase const*> currentMatches;
            for( auto const& test : testCases )
                if( isThrowSafe( test, config ) && filter.matches( test ) )
                    currentMatches.emplace_back( &test );
            return FilterMatch{ filter.name(), currentMatches };
        } );
        return matches;
    }

    const TestSpec::vectorStrings& TestSpec::getInvalidArgs() const{
        return  (m_invalidArgs);
    }

}
// end catch_test_spec.cpp
// start catch_test_spec_parser.cpp

namespace Catch {

    TestSpecParser::TestSpecParser( ITagAliasRegistry const& tagAliases ) : m_tagAliases( &tagAliases
) {}

    TestSpecParser& TestSpecParser::parse( std::string const& arg ) {
        m_mode = None;
        m_exclusion = false;
        m_arg = m_tagAliases->expandAliases( arg );
        m_escapeChars.clear();
        m_substring.reserve(m_arg.size());
        m_patternName.reserve(m_arg.size());
        m_realPatternPos = 0;

        for( m_pos = 0; m_pos < m_arg.size(); ++m_pos )
          //if visitChar fails
           if( !visitChar( m_arg[m_pos] ) ){
               m_testSpec.m_invalidArgs.push_back(arg);
               break;
           }
        endMode();
        return *this;
    }
    TestSpec TestSpecParser::testSpec() {
        addFilter();
        return m_testSpec;
    }
    bool TestSpecParser::visitChar( char c ) {
        if( (m_mode != EscapedName) && (c == '\\') ) {
            escape();
            addCharToPattern(c);
            return true;
        }else if((m_mode != EscapedName) && (c == ',') )  {
            return separate();
        }

        switch( m_mode ) {
        case None:
            if( processNoneChar( c ) )
```

```
                return true;
            break;
        case Name:
            processNameChar( c );
            break;
        case EscapedName:
            endMode();
            addCharToPattern(c);
            return true;
        default:
        case Tag:
        case QuotedName:
            if( processOtherChar( c ) )
                return true;
            break;
        }

        m_substring += c;
        if( !isControlChar( c ) ) {
            m_patternName += c;
            m_realPatternPos++;
        }
        return true;
    }
    // Two of the processing methods return true to signal the caller to return
    // without adding the given character to the current pattern strings
    bool TestSpecParser::processNoneChar( char c ) {
        switch( c ) {
        case ' ':
            return true;
        case '~':
            m_exclusion = true;
            return false;
        case '[':
            startNewMode( Tag );
            return false;
        case '"':
            startNewMode( QuotedName );
            return false;
        default:
            startNewMode( Name );
            return false;
        }
    }
    void TestSpecParser::processNameChar( char c ) {
        if( c == '[' ) {
            if( m_substring == "exclude:" )
                m_exclusion = true;
            else
                endMode();
            startNewMode( Tag );
        }
    }
    bool TestSpecParser::processOtherChar( char c ) {
        if( !isControlChar( c ) )
            return false;
        m_substring += c;
        endMode();
        return true;
    }
    void TestSpecParser::startNewMode( Mode mode ) {
        m_mode = mode;
    }
    void TestSpecParser::endMode() {
        switch( m_mode ) {
        case Name:
        case QuotedName:
            return addNamePattern();
        case Tag:
            return addTagPattern();
        case EscapedName:
            revertBackToLastMode();
            return;
        case None:
        default:
            return startNewMode( None );
        }
    }
    void TestSpecParser::escape() {
        saveLastMode();
        m_mode = EscapedName;
        m_escapeChars.push_back(m_realPatternPos);
    }
    bool TestSpecParser::isControlChar( char c ) const {
        switch( m_mode ) {
            default:
                return false;
            case None:
                return c == '~';
            case Name:
                return c == '[';
            case EscapedName:
                return true;
            case QuotedName:
                return c == '"';
            case Tag:
                return c == '[' || c == ']';
        }
    }

    void TestSpecParser::addFilter() {
        if( !m_currentFilter.m_patterns.empty() ) {
            m_testSpec.m_filters.push_back( m_currentFilter );
            m_currentFilter = TestSpec::Filter();
        }
    }

    void TestSpecParser::saveLastMode() {
      lastMode = m_mode;
    }

    void TestSpecParser::revertBackToLastMode() {
      m_mode = lastMode;
    }

    bool TestSpecParser::separate() {
      if( (m_mode==QuotedName) || (m_mode==Tag) ){
         //invalid argument, signal failure to previous scope.
         m_mode = None;
         m_pos = m_arg.size();
         m_substring.clear();
         m_patternName.clear();
         m_realPatternPos = 0;
         return false;
      }
      endMode();
      addFilter();
      return true; //success
    }

    std::string TestSpecParser::preprocessPattern() {
        std::string token = m_patternName;
        for (std::size_t i = 0; i < m_escapeChars.size(); ++i)
            token = token.substr(0, m_escapeChars[i] - i) + token.substr(m_escapeChars[i] - i + 1);
        m_escapeChars.clear();
        if (startsWith(token, "exclude:")) {
            m_exclusion = true;
            token = token.substr(8);
        }

        m_patternName.clear();
        m_realPatternPos = 0;

        return token;
    }

    void TestSpecParser::addNamePattern() {
        auto token = preprocessPattern();

        if (!token.empty()) {
            TestSpec::PatternPtr pattern = std::make_shared<TestSpec::NamePattern>(token,
      m_substring);
            if (m_exclusion)
                pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
            m_currentFilter.m_patterns.push_back(pattern);
        }
        m_substring.clear();
        m_exclusion = false;
        m_mode = None;
    }

    void TestSpecParser::addTagPattern() {
        auto token = preprocessPattern();

        if (!token.empty()) {
            // If the tag pattern is the "hide and tag" shorthand (e.g. [.foo])
            // we have to create a separate hide tag and shorten the real one
            if (token.size() > 1 && token[0] == '.') {
                token.erase(token.begin());
                TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(".",
      m_substring);
                if (m_exclusion) {
                    pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
                }
                m_currentFilter.m_patterns.push_back(pattern);
            }

            TestSpec::PatternPtr pattern = std::make_shared<TestSpec::TagPattern>(token, m_substring);

            if (m_exclusion) {
                pattern = std::make_shared<TestSpec::ExcludedPattern>(pattern);
            }
            m_currentFilter.m_patterns.push_back(pattern);
        }
        m_substring.clear();
        m_exclusion = false;
        m_mode = None;
    }

    TestSpec parseTestSpec( std::string const& arg ) {
        return TestSpecParser( ITagAliasRegistry::get() ).parse( arg ).testSpec();
    }

} // namespace Catch
// end catch_test_spec_parser.cpp
// start catch_timer.cpp

#include <chrono>

static const uint64_t nanosecondsInSecond = 1000000000;

namespace Catch {

    auto getCurrentNanosecondsSinceEpoch() -> uint64_t {
        return std::chrono::duration_cast<std::chrono::nanoseconds>(
std::chrono::high_resolution_clock::now().time_since_epoch() ).count();
    }

    namespace {
        auto estimateClockResolution() -> uint64_t {
            uint64_t sum = 0;
            static const uint64_t iterations = 1000000;

            auto startTime = getCurrentNanosecondsSinceEpoch();

            for( std::size_t i = 0; i < iterations; ++i ) {

                uint64_t ticks;
                uint64_t baseTicks = getCurrentNanosecondsSinceEpoch();
                do {
                    ticks = getCurrentNanosecondsSinceEpoch();
                } while( ticks == baseTicks );

                auto delta = ticks - baseTicks;
                sum += delta;

                // If we have been calibrating for over 3 seconds -- the clock
                // is terrible and we should move on.
                // TBD: How to signal that the measured resolution is probably wrong?
                if (ticks > startTime + 3 * nanosecondsInSecond) {
                    return sum / ( i + 1u );
                }
            }

            // We're just taking the mean, here. To do better we could take the std. dev and exclude
outliers
            // - and potentially do more iterations if there's a high variance.
            return sum/iterations;
        }
    }
    auto getEstimatedClockResolution() -> uint64_t {
        static auto s_resolution = estimateClockResolution();
        return s_resolution;
    }

    void Timer::start() {
       m_nanoseconds = getCurrentNanosecondsSinceEpoch();
    }
    auto Timer::getElapsedNanoseconds() const -> uint64_t {
        return getCurrentNanosecondsSinceEpoch() - m_nanoseconds;
    }
    auto Timer::getElapsedMicroseconds() const -> uint64_t {
        return getElapsedNanoseconds()/1000;
    }
    auto Timer::getElapsedMilliseconds() const -> unsigned int {
        return static_cast<unsigned int>(getElapsedMicroseconds()/1000);
    }
    auto Timer::getElapsedSeconds() const -> double {
        return getElapsedMicroseconds()/1000000.0;
    }

```

```
} // namespace Catch
// end catch_timer.cpp
// start catch_tostring.cpp

#if defined(__clang__)
#    pragma clang diagnostic push
#    pragma clang diagnostic ignored "-Wexit-time-destructors"
#    pragma clang diagnostic ignored "-Wglobal-constructors"
#endif

// Enable specific decls locally
#if !defined(CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER)
#define CATCH_CONFIG_ENABLE_CHRONO_STRINGMAKER
#endif

#include <cmath>
#include <iomanip>

namespace Catch {

namespace Detail {

    const std::string unprintableString = "{?}";

    namespace {
        const int hexThreshold = 255;

        struct Endianness {
            enum Arch { Big, Little };

            static Arch which() {
                int one = 1;
                // If the lowest byte we read is non-zero, we can assume
                // that little endian format is used.
                auto value = *reinterpret_cast<char*>(&one);
                return value ? Little : Big;
            }
        };
    }

    std::string rawMemoryToString( const void *object, std::size_t size ) {
        // Reverse order for little endian architectures
        int i = 0, end = static_cast<int>( size ), inc = 1;
        if( Endianness::which() == Endianness::Little ) {
            i = end-1;
            end = inc = -1;
        }

        unsigned char const *bytes = static_cast<unsigned char const *>(object);
        ReusableStringStream rss;
        rss « "0x" « std::setfill('0') « std::hex;
        for( ; i != end; i += inc )
             rss « std::setw(2) « static_cast<unsigned>(bytes[i]);
       return rss.str();
    }
}

template<typename T>
std::string fpToString( T value, int precision ) {
    if (Catch::isnan(value)) {
        return "nan";
    }

    ReusableStringStream rss;
    rss « std::setprecision( precision )
        « std::fixed
        « value;
    std::string d = rss.str();
    std::size_t i = d.find_last_not_of( '0' );
    if( i != std::string::npos && i != d.size()-1 ) {
        if( d[i] == '.' )
            i++;
        d = d.substr( 0, i+1 );
    }
    return d;
}

//
//   Out-of-line defs for full specialization of StringMaker
//

std::string StringMaker<std::string>::convert(const std::string& str) {
    if (!getCurrentContext().getConfig()->showInvisibles()) {
        return '"' + str + '"';
    }

    std::string s("\"");
    for (char c : str) {
        switch (c) {
        case '\n':
            s.append("\\n");
            break;
        case '\t':
            s.append("\\t");
            break;
        default:
            s.push_back(c);
            break;
        }
    }
    s.append("\"");
    return s;
}

#ifdef CATCH_CONFIG_CPP17_STRING_VIEW
std::string StringMaker<std::string_view>::convert(std::string_view str) {
    return ::Catch::Detail::stringify(std::string{ str });
}
#endif

std::string StringMaker<char const*>::convert(char const* str) {
    if (str) {
        return ::Catch::Detail::stringify(std::string{ str });
    } else {
        return{ "{null string}" };
    }
}
std::string StringMaker<char*>::convert(char* str) {
    if (str) {
        return ::Catch::Detail::stringify(std::string{ str });
    } else {
        return{ "{null string}" };
    }
}

#ifdef CATCH_CONFIG_WCHAR
std::string StringMaker<std::wstring>::convert(const std::wstring& wstr) {
    std::string s;
    s.reserve(wstr.size());
    for (auto c : wstr) {
        s += (c <= 0xff) ? static_cast<char>(c) : '?';
    }
    return ::Catch::Detail::stringify(s);
}

# ifdef CATCH_CONFIG_CPP17_STRING_VIEW
std::string StringMaker<std::wstring_view>::convert(std::wstring_view str) {
    return StringMaker<std::wstring>::convert(std::wstring(str));
}
# endif

std::string StringMaker<wchar_t const*>::convert(wchar_t const * str) {
    if (str) {
        return ::Catch::Detail::stringify(std::wstring{ str });
    } else {
        return{ "{null string}" };
    }
}
std::string StringMaker<wchar_t *>::convert(wchar_t * str) {
    if (str) {
        return ::Catch::Detail::stringify(std::wstring{ str });
    } else {
        return{ "{null string}" };
    }
}
#endif

#if defined(CATCH_CONFIG_CPP17_BYTE)
#include <cstddef>
std::string StringMaker<std::byte>::convert(std::byte value) {
    return ::Catch::Detail::stringify(std::to_integer<unsigned long long>(value));
}
#endif // defined(CATCH_CONFIG_CPP17_BYTE)

std::string StringMaker<int>::convert(int value) {
    return ::Catch::Detail::stringify(static_cast<long long>(value));
}
std::string StringMaker<long>::convert(long value) {
    return ::Catch::Detail::stringify(static_cast<long long>(value));
}
std::string StringMaker<long long>::convert(long long value) {
    ReusableStringStream rss;
    rss « value;
    if (value > Detail::hexThreshold) {
        rss « " (0x" « std::hex « value « ')';
    }
    return rss.str();
}

std::string StringMaker<unsigned int>::convert(unsigned int value) {
    return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
}
std::string StringMaker<unsigned long>::convert(unsigned long value) {
    return ::Catch::Detail::stringify(static_cast<unsigned long long>(value));
}
std::string StringMaker<unsigned long long>::convert(unsigned long long value) {
    ReusableStringStream rss;
    rss « value;
    if (value > Detail::hexThreshold) {
        rss « " (0x" « std::hex « value « ')';
    }
    return rss.str();
}

std::string StringMaker<bool>::convert(bool b) {
    return b ? "true" : "false";
}

std::string StringMaker<signed char>::convert(signed char value) {
    if (value == '\r') {
        return "'\\r'";
    } else if (value == '\f') {
        return "'\\f'";
    } else if (value == '\n') {
        return "'\\n'";
    } else if (value == '\t') {
        return "'\\t'";
    } else if ('\0' <= value && value < ' ') {
        return ::Catch::Detail::stringify(static_cast<unsigned int>(value));
    } else {
        char chstr[] = "' '";
        chstr[1] = value;
        return chstr;
    }
}
std::string StringMaker<char>::convert(char c) {
    return ::Catch::Detail::stringify(static_cast<signed char>(c));
}
std::string StringMaker<unsigned char>::convert(unsigned char c) {
    return ::Catch::Detail::stringify(static_cast<char>(c));
}

std::string StringMaker<std::nullptr_t>::convert(std::nullptr_t) {
    return "nullptr";
}

int StringMaker<float>::precision = 5;

std::string StringMaker<float>::convert(float value) {
    return fpToString(value, precision) + 'f';
}

int StringMaker<double>::precision = 10;

std::string StringMaker<double>::convert(double value) {
    return fpToString(value, precision);
}

std::string ratio_string<std::atto>::symbol() { return "a"; }
std::string ratio_string<std::femto>::symbol() { return "f"; }
std::string ratio_string<std::pico>::symbol() { return "p"; }
std::string ratio_string<std::nano>::symbol() { return "n"; }
std::string ratio_string<std::micro>::symbol() { return "u"; }
std::string ratio_string<std::milli>::symbol() { return "m"; }

} // end namespace Catch

#if defined(__clang__)
#    pragma clang diagnostic pop
#endif

// end catch_tostring.cpp
// start catch_totals.cpp

namespace Catch {

    Counts Counts::operator - ( Counts const& other ) const {
        Counts diff;
        diff.passed = passed - other.passed;
        diff.failed = failed - other.failed;
        diff.failedButOk = failedButOk - other.failedButOk;
        return diff;
    }

    Counts& Counts::operator += ( Counts const& other ) {
        passed += other.passed;
        failed += other.failed;
        failedButOk += other.failedButOk;
        return *this;
    }

    std::size_t Counts::total() const {
        return passed + failed + failedButOk;
    }
    bool Counts::allPassed() const {
        return failed == 0 && failedButOk == 0;
    }
    bool Counts::allOk() const {
        return failed == 0;
    }

    Totals Totals::operator - ( Totals const& other ) const {
        Totals diff;
        diff.assertions = assertions - other.assertions;
        diff.testCases = testCases - other.testCases;
        return diff;
    }

    Totals& Totals::operator += ( Totals const& other ) {
        assertions += other.assertions;
        testCases += other.testCases;
        return *this;
    }

    Totals Totals::delta( Totals const& prevTotals ) const {
        Totals diff = *this - prevTotals;
        if( diff.assertions.failed > 0 )
            ++diff.testCases.failed;
        else if( diff.assertions.failedButOk > 0 )
            ++diff.testCases.failedButOk;
        else
            ++diff.testCases.passed;
        return diff;
    }

}
// end catch_totals.cpp
// start catch_uncaught_exceptions.cpp

// start catch_config_uncaught_exceptions.hpp

//              Copyright Catch2 Authors
// Distributed under the Boost Software License, Version 1.0.
//   (See accompanying file LICENSE_1_0.txt or copy at
//        https://www.boost.org/LICENSE_1_0.txt)

// SPDX-License-Identifier: BSL-1.0

#ifndef CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
#define CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP

#if defined(_MSC_VER)
#  if _MSC_VER >= 1900 // Visual Studio 2015 or newer
#    define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
#  endif
#endif

#include <exception>

#if defined(__cpp_lib_uncaught_exceptions) \
    && !defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)

#  define CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
#endif // __cpp_lib_uncaught_exceptions

#if defined(CATCH_INTERNAL_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS) \
    && !defined(CATCH_CONFIG_NO_CPP17_UNCAUGHT_EXCEPTIONS) \
    && !defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)

#  define CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS
#endif

#endif // CATCH_CONFIG_UNCAUGHT_EXCEPTIONS_HPP
// end catch_config_uncaught_exceptions.hpp
#include <exception>

namespace Catch {
    bool uncaught_exceptions() {
```

```
#if defined(CATCH_CONFIG_DISABLE_EXCEPTIONS)
        return false;
#elif defined(CATCH_CONFIG_CPP17_UNCAUGHT_EXCEPTIONS)
        return std::uncaught_exceptions() > 0;
#else
        return std::uncaught_exception();
#endif
  }
} // end namespace Catch
// end catch_uncaught_exceptions.cpp
// start catch_version.cpp

#include <ostream>

namespace Catch {

    Version::Version
        (   unsigned int _majorVersion,
            unsigned int _minorVersion,
            unsigned int _patchNumber,
            char const * const _branchName,
            unsigned int _buildNumber )
    :   majorVersion( _majorVersion ),
        minorVersion( _minorVersion ),
        patchNumber( _patchNumber ),
        branchName( _branchName ),
        buildNumber( _buildNumber )
    {}

    std::ostream& operator « ( std::ostream& os, Version const& version ) {
        os  « version.majorVersion « '.'
            « version.minorVersion « '.'
            « version.patchNumber;
        // branchName is never null -> 0th char is \0 if it is empty
        if (version.branchName[0]) {
            os « '-' « version.branchName
               « '.' « version.buildNumber;
        }
        return os;
    }

    Version const& libraryVersion() {
        static Version version( 2, 13, 10, "", 0 );
        return version;
    }

}
// end catch_version.cpp
// start catch_wildcard_pattern.cpp

namespace Catch {

    WildcardPattern::WildcardPattern( std::string const& pattern,
                                      CaseSensitive::Choice caseSensitivity )
    :   m_caseSensitivity( caseSensitivity ),
        m_pattern( normaliseString( pattern ) )
    {
        if( startsWith( m_pattern, '*' ) ) {
            m_pattern = m_pattern.substr( 1 );
            m_wildcard = WildcardAtStart;
        }
        if( endsWith( m_pattern, '*' ) ) {
            m_pattern = m_pattern.substr( 0, m_pattern.size()-1 );
            m_wildcard = static_cast<WildcardPosition>( m_wildcard | WildcardAtEnd );
        }
    }

    bool WildcardPattern::matches( std::string const& str ) const {
        switch( m_wildcard ) {
            case NoWildcard:
                return m_pattern == normaliseString( str );
            case WildcardAtStart:
                return endsWith( normaliseString( str ), m_pattern );
            case WildcardAtEnd:
                return startsWith( normaliseString( str ), m_pattern );
            case WildcardAtBothEnds:
                return contains( normaliseString( str ), m_pattern );
            default:
                CATCH_INTERNAL_ERROR( "Unknown enum" );
        }
    }

    std::string WildcardPattern::normaliseString( std::string const& str ) const {
        return trim( m_caseSensitivity == CaseSensitive::No ? toLower( str ) : str );
    }
}
// end catch_wildcard_pattern.cpp
// start catch_xmlwriter.cpp

#include <iomanip>
#include <type_traits>

namespace Catch {

namespace {

    size_t trailingBytes(unsigned char c) {
        if ((c & 0xE0) == 0xC0) {
            return 2;
        }
        if ((c & 0xF0) == 0xE0) {
            return 3;
        }
        if ((c & 0xF8) == 0xF0) {
            return 4;
        }
        CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
    }

    uint32_t headerValue(unsigned char c) {
        if ((c & 0xE0) == 0xC0) {
            return c & 0x1F;
        }
        if ((c & 0xF0) == 0xE0) {
            return c & 0x0F;
        }
        if ((c & 0xF8) == 0xF0) {
            return c & 0x07;
        }
        CATCH_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
    }

    void hexEscapeChar(std::ostream& os, unsigned char c) {
        std::ios_base::fmtflags f(os.flags());
        os « "\\x"
            « std::uppercase « std::hex « std::setfill('0') « std::setw(2)
            « static_cast<int>(c);
        os.flags(f);
    }

    bool shouldNewline(XmlFormatting fmt) {
        return !!(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
XmlFormatting::Newline));
    }

    bool shouldIndent(XmlFormatting fmt) {
        return !!(static_cast<std::underlying_type<XmlFormatting>::type>(fmt &
XmlFormatting::Indent));
    }

} // anonymous namespace

    XmlFormatting operator | (XmlFormatting lhs, XmlFormatting rhs) {
        return static_cast<XmlFormatting>(
            static_cast<std::underlying_type<XmlFormatting>::type>(lhs) |
            static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
        );
    }

    XmlFormatting operator & (XmlFormatting lhs, XmlFormatting rhs) {
        return static_cast<XmlFormatting>(
            static_cast<std::underlying_type<XmlFormatting>::type>(lhs) &
            static_cast<std::underlying_type<XmlFormatting>::type>(rhs)
        );
    }

    XmlEncode::XmlEncode( std::string const& str, ForWhat forWhat )
    :   m_str( str ),
        m_forWhat( forWhat )
    {}

    void XmlEncode::encodeTo( std::ostream& os ) const {
        // Apostrophe escaping not necessary if we always use " to write attributes
        // (see: http://www.w3.org/TR/xml/#syntax)

        for( std::size_t idx = 0; idx < m_str.size(); ++ idx ) {
            unsigned char c = m_str[idx];
            switch (c) {
            case '<':   os « "&lt;"; break;
            case '&':   os « "&amp;"; break;

            case '>':
                // See: http://www.w3.org/TR/xml/#syntax
                if (idx > 2 && m_str[idx - 1] == ']' && m_str[idx - 2] == ']')
```

```
                    os « "&gt;";
                else
                    os « c;
                break;

            case '\"':
                if (m_forWhat == ForAttributes)
                    os « "&quot;";
                else
                    os « c;
                break;

            default:
                // Check for control characters and invalid utf-8

                // Escape control characters in standard ascii
                // see http://stackoverflow.com/questions/404107/why-are-control-characters-illegal-in-xml-1-0
                if (c < 0x09 || (c > 0x0D && c < 0x20) || c == 0x7F) {
                    hexEscapeChar(os, c);
                    break;
                }

                // Plain ASCII: Write it to stream
                if (c < 0x7F) {
                    os « c;
                    break;
                }

                // UTF-8 territory
                // Check if the encoding is valid and if it is not, hex escape bytes.
                // Important: We do not check the exact decoded values for validity, only the encoding format
                // First check that this bytes is a valid lead byte:
                // This means that it is not encoded as 1111 1XXX
                // Or as 10XX XXXX
                if (c <  0xC0 ||
                    c >= 0xF8) {
                    hexEscapeChar(os, c);
                    break;
                }

                auto encBytes = trailingBytes(c);
                // Are there enough bytes left to avoid accessing out-of-bounds memory?
                if (idx + encBytes - 1 >= m_str.size()) {
                    hexEscapeChar(os, c);
                    break;
                }
                // The header is valid, check data
                // The next encBytes bytes must together be a valid utf-8
                // This means: bitpattern 10XX XXXX and the extracted value is sane (ish)
                bool valid = true;
                uint32_t value = headerValue(c);
                for (std::size_t n = 1; n < encBytes; ++n) {
                    unsigned char nc = m_str[idx + n];
                    valid &= ((nc & 0xC0) == 0x80);
                    value = (value « 6) | (nc & 0x3F);
                }

                if (
                    // Wrong bit pattern of following bytes
                    (!valid) ||
                    // Overlong encodings
                    (value < 0x80) ||
                    (0x80 <= value && value < 0x800   && encBytes > 2) ||
                    (0x800 < value && value < 0x10000 && encBytes > 3) ||
                    // Encoded value out of range
                    (value >= 0x110000)
                    ) {
                    hexEscapeChar(os, c);
                    break;
                }

                // If we got here, this is in fact a valid(ish) utf-8 sequence
                for (std::size_t n = 0; n < encBytes; ++n) {
                    os « m_str[idx + n];
                }
                idx += encBytes - 1;
                break;
            }
        }
    }

    std::ostream& operator « ( std::ostream& os, XmlEncode const& xmlEncode ) {
        xmlEncode.encodeTo( os );
        return os;
    }

    XmlWriter::ScopedElement::ScopedElement( XmlWriter* writer, XmlFormatting fmt )
    :   m_writer( writer ),
        m_fmt(fmt)
    {}

    XmlWriter::ScopedElement::ScopedElement( ScopedElement&& other ) noexcept
    :   m_writer( other.m_writer ),
        m_fmt(other.m_fmt)
    {
        other.m_writer = nullptr;
        other.m_fmt = XmlFormatting::None;
    }
    XmlWriter::ScopedElement& XmlWriter::ScopedElement::operator=( ScopedElement&& other ) noexcept {
        if ( m_writer ) {
            m_writer->endElement();
        }
        m_writer = other.m_writer;
        other.m_writer = nullptr;
        m_fmt = other.m_fmt;
        other.m_fmt = XmlFormatting::None;
        return *this;
    }

    XmlWriter::ScopedElement::~ScopedElement() {
        if (m_writer) {
            m_writer->endElement(m_fmt);
        }
    }

    XmlWriter::ScopedElement& XmlWriter::ScopedElement::writeText( std::string const& text,
XmlFormatting fmt ) {
        m_writer->writeText( text, fmt );
        return *this;
    }

    XmlWriter::XmlWriter( std::ostream& os ) : m_os( os )
    {
        writeDeclaration();
    }

    XmlWriter::~XmlWriter() {
        while (!m_tags.empty()) {
            endElement();
        }
        newlineIfNecessary();
    }

    XmlWriter& XmlWriter::startElement( std::string const& name, XmlFormatting fmt ) {
        ensureTagClosed();
        newlineIfNecessary();
        if (shouldIndent(fmt)) {
            m_os « m_indent;
            m_indent += "  ";
        }
        m_os « '<' « name;
        m_tags.push_back( name );
        m_tagIsOpen = true;
        applyFormatting(fmt);
        return *this;
    }

    XmlWriter::ScopedElement XmlWriter::scopedElement( std::string const& name, XmlFormatting fmt ) {
        ScopedElement scoped( this, fmt );
        startElement( name, fmt );
        return scoped;
    }

    XmlWriter& XmlWriter::endElement(XmlFormatting fmt) {
        m_indent = m_indent.substr(0, m_indent.size() - 2);

        if( m_tagIsOpen ) {
            m_os « "/>";
            m_tagIsOpen = false;
        } else {
            newlineIfNecessary();
            if (shouldIndent(fmt)) {
                m_os « m_indent;
            }
            m_os « "</" « m_tags.back() « ">";
        }
        m_os « std::flush;
        applyFormatting(fmt);
        m_tags.pop_back();
        return *this;
    }

    XmlWriter& XmlWriter::writeAttribute( std::string const& name, std::string const& attribute ) {
        if( !name.empty() && !attribute.empty() )
            m_os « ' ' « name « "=\"" « XmlEncode( attribute, XmlEncode::ForAttributes ) « '"';
        return *this;
    }

    XmlWriter& XmlWriter::writeAttribute( std::string const& name, bool attribute ) {
        m_os « ' ' « name « "=\"" « ( attribute ? "true" : "false" ) « '"';
        return *this;
    }

    XmlWriter& XmlWriter::writeText( std::string const& text, XmlFormatting fmt) {
        if( !text.empty() ){
            bool tagWasOpen = m_tagIsOpen;
            ensureTagClosed();
            if (tagWasOpen && shouldIndent(fmt)) {
                m_os « m_indent;
            }
            m_os « XmlEncode( text );
            applyFormatting(fmt);
        }
        return *this;
    }

    XmlWriter& XmlWriter::writeComment( std::string const& text, XmlFormatting fmt) {
        ensureTagClosed();
        if (shouldIndent(fmt)) {
            m_os « m_indent;
        }
        m_os « "<!--" « text « "-->";
        applyFormatting(fmt);
        return *this;
    }

    void XmlWriter::writeStylesheetRef( std::string const& url ) {
        m_os « "<?xml-stylesheet type=\"text/xsl\" href=\"" « url « "\"?>\n";
    }

    XmlWriter& XmlWriter::writeBlankLine() {
        ensureTagClosed();
        m_os « '\n';
        return *this;
    }

    void XmlWriter::ensureTagClosed() {
        if( m_tagIsOpen ) {
            m_os « '>' « std::flush;
            newlineIfNecessary();
            m_tagIsOpen = false;
        }
    }

    void XmlWriter::applyFormatting(XmlFormatting fmt) {
        m_needsNewline = shouldNewline(fmt);
    }

    void XmlWriter::writeDeclaration() {
        m_os « "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
    }

    void XmlWriter::newlineIfNecessary() {
        if( m_needsNewline ) {
            m_os « std::endl;
            m_needsNewline = false;
        }
    }
}
// end catch_xmlwriter.cpp
// start catch_reporter_bases.cpp

#include <cstring>
#include <cfloat>
#include <cstdio>
#include <cassert>
#include <memory>

namespace Catch {
    void prepareExpandedExpression(AssertionResult& result) {
        result.getExpandedExpression();
    }

    // Because formatting using c++ streams is stateful, drop down to C is required
    // Alternatively we could use stringstream, but its performance is... not good.
    std::string getFormattedDuration( double duration ) {
        // Max exponent + 1 is required to represent the whole part
        // + 1 for decimal point
        // + 3 for the 3 decimal places
        // + 1 for null terminator
        const std::size_t maxDoubleSize = DBL_MAX_10_EXP + 1 + 1 + 3 + 1;
        char buffer[maxDoubleSize];

        // Save previous errno, to prevent sprintf from overwriting it
        ErrnoGuard guard;
#ifdef _MSC_VER
        sprintf_s(buffer, "%.3f", duration);
#else
        std::sprintf(buffer, "%.3f", duration);
#endif
        return std::string(buffer);
    }

    bool shouldShowDuration( IConfig const& config, double duration ) {
        if ( config.showDurations() == ShowDurations::Always ) {
            return true;
        }
        if ( config.showDurations() == ShowDurations::Never ) {
            return false;
        }
        const double min = config.minDuration();
        return min >= 0 && duration >= min;
    }

    std::string serializeFilters( std::vector<std::string> const& container ) {
        ReusableStringStream oss;
        bool first = true;
        for (auto&& filter : container)
        {
            if (!first)
                oss « ' ';
            else
                first = false;

            oss « filter;
        }
        return oss.str();
    }

    TestEventListenerBase::TestEventListenerBase(ReporterConfig const & _config)
        :StreamingReporterBase(_config) {}

    std::set<Verbosity> TestEventListenerBase::getSupportedVerbosities() {
        return { Verbosity::Quiet, Verbosity::Normal, Verbosity::High };
    }

    void TestEventListenerBase::assertionStarting(AssertionInfo const &) {}

    bool TestEventListenerBase::assertionEnded(AssertionStats const &) {
        return false;
    }

} // end namespace Catch
// end catch_reporter_bases.cpp
// start catch_reporter_compact.cpp

namespace {

#ifdef CATCH_PLATFORM_MAC
    const char* failedString() { return "FAILED"; }
    const char* passedString() { return "PASSED"; }
#else
    const char* failedString() { return "failed"; }
    const char* passedString() { return "passed"; }
#endif

    // Colour::LightGrey
    Catch::Colour::Code dimColour() { return Catch::Colour::FileName; }

    std::string bothOrAll( std::size_t count ) {
        return count == 1 ? std::string() :
               count == 2 ? "both " : "all " ;
    }

} // anon namespace

namespace Catch {
namespace {
// Colour, message variants:
// - white: No tests ran.
// -   red: Failed [both/all] N test cases, failed [both/all] M assertions.
// - white: Passed [both/all] N test cases (no assertions).
// -   red: Failed N tests cases, failed M assertions.
// - green: Passed [both/all] N tests cases with M assertions.
void printTotals(std::ostream& out, const Totals& totals) {
    if (totals.testCases.total() == 0) {
        out « "No tests ran.";
    } else if (totals.testCases.failed == totals.testCases.total()) {
        Colour colour(Colour::ResultError);
        const std::string qualify_assertions_failed =
            totals.assertions.failed == totals.assertions.total() ?
            bothOrAll(totals.assertions.failed) : std::string();
        out «
            "Failed " « bothOrAll(totals.testCases.failed)
            « pluralise(totals.testCases.failed, "test case") « ", "
            "failed " « qualify_assertions_failed «
            pluralise(totals.assertions.failed, "assertion") « '.';
    } else if (totals.assertions.total() == 0) {
        out «
            "Passed " « bothOrAll(totals.testCases.total())
            « pluralise(totals.testCases.total(), "test case")
            « " (no assertions).";
    } else if (totals.assertions.failed) {
        Colour colour(Colour::ResultError);
        out «
            "Failed " « pluralise(totals.testCases.failed, "test case") « ", "
            "failed " « pluralise(totals.assertions.failed, "assertion") « '.';
    } else {
        Colour colour(Colour::ResultSuccess);
        out «
            "Passed " « bothOrAll(totals.testCases.passed)
            « pluralise(totals.testCases.passed, "test case") «
            " with " « pluralise(totals.assertions.passed, "assertion") « '.';
    }
}

// Implementation of CompactReporter formatting
class AssertionPrinter {
public:
    AssertionPrinter& operator= (AssertionPrinter const&) = delete;
    AssertionPrinter(AssertionPrinter const&) = delete;
    AssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool _printInfoMessages)
        : stream(_stream)
        , result(_stats.assertionResult)
        , messages(_stats.infoMessages)
        , itMessage(_stats.infoMessages.begin())
        , printInfoMessages(_printInfoMessages) {}

    void print() {
        printSourceInfo();

        itMessage = messages.begin();

        switch (result.getResultType()) {
        case ResultWas::Ok:
            printResultType(Colour::ResultSuccess, passedString());
            printOriginalExpression();
            printReconstructedExpression();
            if (!result.hasExpression())
                printRemainingMessages(Colour::None);
            else
                printRemainingMessages();
            break;
        case ResultWas::ExpressionFailed:
            if (result.isOk())
                printResultType(Colour::ResultSuccess, failedString() + std::string(" - but was ok"));
            else
                printResultType(Colour::Error, failedString());
            printOriginalExpression();
            printReconstructedExpression();
            printRemainingMessages();
            break;
        case ResultWas::ThrewException:
            printResultType(Colour::Error, failedString());
            printIssue("unexpected exception with message:");
            printMessage();
            printExpressionWas();
            printRemainingMessages();
            break;
        case ResultWas::FatalErrorCondition:
            printResultType(Colour::Error, failedString());
            printIssue("fatal error condition with message:");
            printMessage();
            printExpressionWas();
            printRemainingMessages();
            break;
        case ResultWas::DidntThrowException:
            printResultType(Colour::Error, failedString());
            printIssue("expected exception, got none");
            printExpressionWas();
            printRemainingMessages();
            break;
        case ResultWas::Info:
```

```
            printResultType(Colour::None, "info");
            printMessage();
            printRemainingMessages();
            break;
        case ResultWas::Warning:
            printResultType(Colour::None, "warning");
            printMessage();
            printRemainingMessages();
            break;
        case ResultWas::ExplicitFailure:
            printResultType(Colour::Error, failedString());
            printIssue("explicitly");
            printRemainingMessages(Colour::None);
            break;
            // These cases are here to prevent compiler warnings
        case ResultWas::Unknown:
        case ResultWas::FailureBit:
        case ResultWas::Exception:
            printResultType(Colour::Error, "** internal error **");
            break;
        }
    }

private:
    void printSourceInfo() const {
        Colour colourGuard(Colour::FileName);
        stream « result.getSourceInfo() « ':';
    }

    void printResultType(Colour::Code colour, std::string const& passOrFail) const {
        if (!passOrFail.empty()) {
            {
                Colour colourGuard(colour);
                stream « ' ' « passOrFail;
            }
            stream « ':';
        }
    }

    void printIssue(std::string const& issue) const {
        stream « ' ' « issue;
    }

    void printExpressionWas() {
        if (result.hasExpression()) {
            stream « ';';
            {
                Colour colour(dimColour());
                stream « " expression was:";
            }
            printOriginalExpression();
        }
    }

    void printOriginalExpression() const {
        if (result.hasExpression()) {
            stream « ' ' « result.getExpression();
        }
    }

    void printReconstructedExpression() const {
        if (result.hasExpandedExpression()) {
            {
                Colour colour(dimColour());
                stream « " for: ";
            }
            stream « result.getExpandedExpression();
        }
    }

    void printMessage() {
        if (itMessage != messages.end()) {
            stream « " '" « itMessage->message « '\'';
            ++itMessage;
        }
    }

    void printRemainingMessages(Colour::Code colour = dimColour()) {
        if (itMessage == messages.end())
            return;

        const auto itEnd = messages.cend();
        const auto N = static_cast<std::size_t>(std::distance(itMessage, itEnd));

        {
            Colour colourGuard(colour);
            stream « " with " « pluralise(N, "message") « ':';
        }

        while (itMessage != itEnd) {
            // If this assertion is a warning ignore any INFO messages
            if (printInfoMessages || itMessage->type != ResultWas::Info) {
                printMessage();
                if (itMessage != itEnd) {
                    Colour colourGuard(dimColour());
                    stream « " and";
                }
                continue;
            }
            ++itMessage;
        }
    }

private:
    std::ostream& stream;
    AssertionResult const& result;
    std::vector<MessageInfo> messages;
    std::vector<MessageInfo>::const_iterator itMessage;
    bool printInfoMessages;
};

} // anon namespace

        std::string CompactReporter::getDescription() {
            return "Reports test results on a single line, suitable for IDEs";
        }

        void CompactReporter::noMatchingTestCases( std::string const& spec ) {
            stream « "No test cases matched '" « spec « '\'' « std::endl;
        }

        void CompactReporter::assertionStarting( AssertionInfo const& ) {}

        bool CompactReporter::assertionEnded( AssertionStats const& _assertionStats ) {
            AssertionResult const& result = _assertionStats.assertionResult;

            bool printInfoMessages = true;

            // Drop out if result was successful and we're not printing those
            if( !m_config->includeSuccessfulResults() && result.isOk() ) {
                if( result.getResultType() != ResultWas::Warning )
                    return false;
                printInfoMessages = false;
            }

            AssertionPrinter printer( stream, _assertionStats, printInfoMessages );
            printer.print();

            stream « std::endl;
            return true;
        }

        void CompactReporter::sectionEnded(SectionStats const& _sectionStats) {
            double dur = _sectionStats.durationInSeconds;
            if ( shouldShowDuration( *m_config, dur ) ) {
                stream « getFormattedDuration( dur ) « " s: " « _sectionStats.sectionInfo.name «
std::endl;
            }
        }

        void CompactReporter::testRunEnded( TestRunStats const& _testRunStats ) {
            printTotals( stream, _testRunStats.totals );
            stream « '\n' « std::endl;
            StreamingReporterBase::testRunEnded( _testRunStats );
        }

        CompactReporter::~CompactReporter() {}

    CATCH_REGISTER_REPORTER( "compact", CompactReporter )

} // end namespace Catch
// end catch_reporter_compact.cpp
// start catch_reporter_console.cpp

#include <cfloat>
#include <cstdio>

#if defined(_MSC_VER)
#pragma warning(push)
#pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
 // Note that 4062 (not all labels are handled and default is missing) is enabled
#endif

#if defined(__clang__)
```

```
#  pragma clang diagnostic push
// For simplicity, benchmarking-only helpers are always enabled
#  pragma clang diagnostic ignored "-Wunused-function"
#endif

namespace Catch {

namespace {

// Formatter impl for ConsoleReporter
class ConsoleAssertionPrinter {
public:
    ConsoleAssertionPrinter& operator= (ConsoleAssertionPrinter const&) = delete;
    ConsoleAssertionPrinter(ConsoleAssertionPrinter const&) = delete;
    ConsoleAssertionPrinter(std::ostream& _stream, AssertionStats const& _stats, bool
_printInfoMessages)
        : stream(_stream),
        stats(_stats),
        result(_stats.assertionResult),
        colour(Colour::None),
        message(result.getMessage()),
        messages(_stats.infoMessages),
        printInfoMessages(_printInfoMessages) {
        switch (result.getResultType()) {
        case ResultWas::Ok:
            colour = Colour::Success;
            passOrFail = "PASSED";
            //if( result.hasMessage() )
            if (_stats.infoMessages.size() == 1)
                messageLabel = "with message";
            if (_stats.infoMessages.size() > 1)
                messageLabel = "with messages";
            break;
        case ResultWas::ExpressionFailed:
            if (result.isOk()) {
                colour = Colour::Success;
                passOrFail = "FAILED - but was ok";
            } else {
                colour = Colour::Error;
                passOrFail = "FAILED";
            }
            if (_stats.infoMessages.size() == 1)
                messageLabel = "with message";
            if (_stats.infoMessages.size() > 1)
                messageLabel = "with messages";
            break;
        case ResultWas::ThrewException:
            colour = Colour::Error;
            passOrFail = "FAILED";
            messageLabel = "due to unexpected exception with ";
            if (_stats.infoMessages.size() == 1)
                messageLabel += "message";
            if (_stats.infoMessages.size() > 1)
                messageLabel += "messages";
            break;
        case ResultWas::FatalErrorCondition:
            colour = Colour::Error;
            passOrFail = "FAILED";
            messageLabel = "due to a fatal error condition";
            break;
        case ResultWas::DidntThrowException:
            colour = Colour::Error;
            passOrFail = "FAILED";
            messageLabel = "because no exception was thrown where one was expected";
            break;
        case ResultWas::Info:
            messageLabel = "info";
            break;
        case ResultWas::Warning:
            messageLabel = "warning";
            break;
        case ResultWas::ExplicitFailure:
            passOrFail = "FAILED";
            colour = Colour::Error;
            if (_stats.infoMessages.size() == 1)
                messageLabel = "explicitly with message";
            if (_stats.infoMessages.size() > 1)
                messageLabel = "explicitly with messages";
            break;
            // These cases are here to prevent compiler warnings
        case ResultWas::Unknown:
        case ResultWas::FailureBit:
        case ResultWas::Exception:
            passOrFail = "** internal error **";
            colour = Colour::Error;
            break;
        }
    }

    void print() const {
        printSourceInfo();
        if (stats.totals.assertions.total() > 0) {
            printResultType();
            printOriginalExpression();
            printReconstructedExpression();
        } else {
            stream « '\n';
        }
        printMessage();
    }

private:
    void printResultType() const {
        if (!passOrFail.empty()) {
            Colour colourGuard(colour);
            stream « passOrFail « ":\n";
        }
    }
    void printOriginalExpression() const {
        if (result.hasExpression()) {
            Colour colourGuard(Colour::OriginalExpression);
            stream « "  ";
            stream « result.getExpressionInMacro();
            stream « '\n';
        }
    }
    void printReconstructedExpression() const {
        if (result.hasExpandedExpression()) {
            stream « "with expansion:\n";
            Colour colourGuard(Colour::ReconstructedExpression);
            stream « Column(result.getExpandedExpression()).indent(2) « '\n';
        }
    }
    void printMessage() const {
        if (!messageLabel.empty())
            stream « messageLabel « ':' « '\n';
        for (auto const& msg : messages) {
            // If this assertion is a warning ignore any INFO messages
            if (printInfoMessages || msg.type != ResultWas::Info)
                stream « Column(msg.message).indent(2) « '\n';
        }
    }
    void printSourceInfo() const {
        Colour colourGuard(Colour::FileName);
        stream « result.getSourceInfo() « ": ";
    }

    std::ostream& stream;
    AssertionStats const& stats;
    AssertionResult const& result;
    Colour::Code colour;
    std::string passOrFail;
    std::string messageLabel;
    std::string message;
    std::vector<MessageInfo> messages;
    bool printInfoMessages;
};

std::size_t makeRatio(std::size_t number, std::size_t total) {
    std::size_t ratio = total > 0 ? CATCH_CONFIG_CONSOLE_WIDTH * number / total : 0;
    return (ratio == 0 && number > 0) ? 1 : ratio;
}

std::size_t& findMax(std::size_t& i, std::size_t& j, std::size_t& k) {
    if (i > j && i > k)
        return i;
    else if (j > k)
        return j;
    else
        return k;
}

struct ColumnInfo {
    enum Justification { Left, Right };
    std::string name;
    int width;
    Justification justification;
};
struct ColumnBreak {};
struct RowBreak {};

class Duration {
    enum class Unit {
        Auto,
        Nanoseconds,
        Microseconds,
        Milliseconds,
        Seconds,
        Minutes
    };
    static const uint64_t s_nanosecondsInAMicrosecond = 1000;
    static const uint64_t s_nanosecondsInAMillisecond = 1000 * s_nanosecondsInAMicrosecond;
    static const uint64_t s_nanosecondsInASecond = 1000 * s_nanosecondsInAMillisecond;
    static const uint64_t s_nanosecondsInAMinute = 60 * s_nanosecondsInASecond;

    double m_inNanoseconds;
    Unit m_units;

public:
    explicit Duration(double inNanoseconds, Unit units = Unit::Auto)
        : m_inNanoseconds(inNanoseconds),
        m_units(units) {
        if (m_units == Unit::Auto) {
            if (m_inNanoseconds < s_nanosecondsInAMicrosecond)
                m_units = Unit::Nanoseconds;
            else if (m_inNanoseconds < s_nanosecondsInAMillisecond)
                m_units = Unit::Microseconds;
            else if (m_inNanoseconds < s_nanosecondsInASecond)
                m_units = Unit::Milliseconds;
            else if (m_inNanoseconds < s_nanosecondsInAMinute)
                m_units = Unit::Seconds;
            else
                m_units = Unit::Minutes;
        }

    }

    auto value() const -> double {
        switch (m_units) {
        case Unit::Microseconds:
            return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMicrosecond);
        case Unit::Milliseconds:
            return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMillisecond);
        case Unit::Seconds:
            return m_inNanoseconds / static_cast<double>(s_nanosecondsInASecond);
        case Unit::Minutes:
            return m_inNanoseconds / static_cast<double>(s_nanosecondsInAMinute);
        default:
            return m_inNanoseconds;
        }
    }
    auto unitsAsString() const -> std::string {
        switch (m_units) {
        case Unit::Nanoseconds:
            return "ns";
        case Unit::Microseconds:
            return "us";
        case Unit::Milliseconds:
            return "ms";
        case Unit::Seconds:
            return "s";
        case Unit::Minutes:
            return "m";
        default:
            return "** internal error **";
        }

    }
    friend auto operator « (std::ostream& os, Duration const& duration) -> std::ostream& {
        return os « duration.value() « ' ' « duration.unitsAsString();
    }
};
} // end anon namespace

class TablePrinter {
    std::ostream& m_os;
    std::vector<ColumnInfo> m_columnInfos;
    std::ostringstream m_oss;
    int m_currentColumn = -1;
    bool m_isOpen = false;

public:
    TablePrinter( std::ostream& os, std::vector<ColumnInfo> columnInfos )
    :   m_os( os ),
        m_columnInfos( std::move( columnInfos ) ) {}

    auto columnInfos() const -> std::vector<ColumnInfo> const& {
        return m_columnInfos;
    }

    void open() {
        if (!m_isOpen) {
            m_isOpen = true;
            *this « RowBreak();

            Columns headerCols;
            Spacer spacer(2);
            for (auto const& info : m_columnInfos) {
                headerCols += Column(info.name).width(static_cast<std::size_t>(info.width - 2));
                headerCols += spacer;
            }
            m_os « headerCols « '\n';

            m_os « Catch::getLineOfChars<'-'>() « '\n';
        }
    }
    void close() {
        if (m_isOpen) {
            *this « RowBreak();
            m_os « std::endl;
            m_isOpen = false;
        }
    }

    template<typename T>
    friend TablePrinter& operator « (TablePrinter& tp, T const& value) {
        tp.m_oss « value;
        return tp;
    }

    friend TablePrinter& operator « (TablePrinter& tp, ColumnBreak) {
        auto colStr = tp.m_oss.str();
        const auto strSize = colStr.size();
        tp.m_oss.str("");
        tp.open();
        if (tp.m_currentColumn == static_cast<int>(tp.m_columnInfos.size() - 1)) {
            tp.m_currentColumn = -1;
            tp.m_os « '\n';
        }
        tp.m_currentColumn++;

        auto colInfo = tp.m_columnInfos[tp.m_currentColumn];
        auto padding = (strSize + 1 < static_cast<std::size_t>(colInfo.width))
            ? std::string(colInfo.width - (strSize + 1), ' ')
            : std::string();
        if (colInfo.justification == ColumnInfo::Left)
            tp.m_os « colStr « padding « ' ';
        else
            tp.m_os « padding « colStr « ' ';
        return tp;
    }

    friend TablePrinter& operator « (TablePrinter& tp, RowBreak) {
        if (tp.m_currentColumn > 0) {
            tp.m_os « '\n';
            tp.m_currentColumn = -1;
        }
        return tp;
    }
};

ConsoleReporter::ConsoleReporter(ReporterConfig const& config)
    : StreamingReporterBase(config),
    m_tablePrinter(new TablePrinter(config.stream(),
        [&config]() -> std::vector<ColumnInfo> {
        if (config.fullConfig()->benchmarkNoAnalysis())
        {
            return{
                { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
                { "     samples", 14, ColumnInfo::Right },
                { "  iterations", 14, ColumnInfo::Right },
                { "        mean", 14, ColumnInfo::Right }
            };
        }
        else
        {
            return{
                { "benchmark name", CATCH_CONFIG_CONSOLE_WIDTH - 43, ColumnInfo::Left },
                { "samples      mean       std dev", 14, ColumnInfo::Right },
                { "iterations   low mean   low std dev", 14, ColumnInfo::Right },
                { "estimated    high mean  high std dev", 14, ColumnInfo::Right }
            };
        }
    }())) {}
ConsoleReporter::~ConsoleReporter() = default;

std::string ConsoleReporter::getDescription() {
    return "Reports test results as plain lines of text";
```

```
}

void ConsoleReporter::noMatchingTestCases(std::string const& spec) {
    stream « "No test cases matched '" « spec « '\" « std::endl;
}

void ConsoleReporter::reportInvalidArguments(std::string const&arg){
    stream « "Invalid Filter: " « arg « std::endl;
}

void ConsoleReporter::assertionStarting(AssertionInfo const&) {}

bool ConsoleReporter::assertionEnded(AssertionStats const& _assertionStats) {
    AssertionResult const& result = _assertionStats.assertionResult;

    bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();

    // Drop out if result was successful but we're not printing them.
    if (!includeResults && result.getResultType() != ResultWas::Warning)
        return false;

    lazyPrint();

    ConsoleAssertionPrinter printer(stream, _assertionStats, includeResults);
    printer.print();
    stream « std::endl;
    return true;
}

void ConsoleReporter::sectionStarting(SectionInfo const& _sectionInfo) {
    m_tablePrinter->close();
    m_headerPrinted = false;
    StreamingReporterBase::sectionStarting(_sectionInfo);
}
void ConsoleReporter::sectionEnded(SectionStats const& _sectionStats) {
    m_tablePrinter->close();
    if (_sectionStats.missingAssertions) {
        lazyPrint();
        Colour colour(Colour::ResultError);
        if (m_sectionStack.size() > 1)
            stream « "\nNo assertions in section";
        else
            stream « "\nNo assertions in test case";
        stream « " '" « _sectionStats.sectionInfo.name « "'\n" « std::endl;
    }
    double dur = _sectionStats.durationInSeconds;
    if (shouldShowDuration(*m_config, dur)) {
        stream « getFormattedDuration(dur) « " s: " « _sectionStats.sectionInfo.name « std::endl;
    }
    if (m_headerPrinted) {
        m_headerPrinted = false;
    }
    StreamingReporterBase::sectionEnded(_sectionStats);
}

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
void ConsoleReporter::benchmarkPreparing(std::string const& name) {
    lazyPrintWithoutClosingBenchmarkTable();

    auto nameCol = Column(name).width(static_cast<std::size_t>(m_tablePrinter->columnInfos()[0].width
- 2));

    bool firstLine = true;
    for (auto line : nameCol) {
        if (!firstLine)
            (*m_tablePrinter) « ColumnBreak() « ColumnBreak() « ColumnBreak();
        else
            firstLine = false;

        (*m_tablePrinter) « line « ColumnBreak();
    }
}

void ConsoleReporter::benchmarkStarting(BenchmarkInfo const& info) {
    (*m_tablePrinter) « info.samples « ColumnBreak()
        « info.iterations « ColumnBreak();
    if (!m_config->benchmarkNoAnalysis())
        (*m_tablePrinter) « Duration(info.estimatedDuration) « ColumnBreak();
}
void ConsoleReporter::benchmarkEnded(BenchmarkStats<> const& stats) {
    if (m_config->benchmarkNoAnalysis())
    {
        (*m_tablePrinter) « Duration(stats.mean.point.count()) « ColumnBreak();
    }
    else
    {
        (*m_tablePrinter) « ColumnBreak()
```

```
            « Duration(stats.mean.point.count()) « ColumnBreak()
            « Duration(stats.mean.lower_bound.count()) « ColumnBreak()
            « Duration(stats.mean.upper_bound.count()) « ColumnBreak() « ColumnBreak()
            « Duration(stats.standardDeviation.point.count()) « ColumnBreak()
            « Duration(stats.standardDeviation.lower_bound.count()) « ColumnBreak()
            « Duration(stats.standardDeviation.upper_bound.count()) « ColumnBreak() « ColumnBreak() «
ColumnBreak() « ColumnBreak() « ColumnBreak();
    }
}

void ConsoleReporter::benchmarkFailed(std::string const& error) {
    Colour colour(Colour::Red);
    (*m_tablePrinter)
        « "Benchmark failed (" « error « ')'
        « ColumnBreak() « RowBreak();
}
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

void ConsoleReporter::testCaseEnded(TestCaseStats const& _testCaseStats) {
    m_tablePrinter->close();
    StreamingReporterBase::testCaseEnded(_testCaseStats);
    m_headerPrinted = false;
}
void ConsoleReporter::testGroupEnded(TestGroupStats const& _testGroupStats) {
    if (currentGroupInfo.used) {
        printSummaryDivider();
        stream « "Summary for group '" « _testGroupStats.groupInfo.name « "':\n";
        printTotals(_testGroupStats.totals);
        stream « '\n' « std::endl;
    }
    StreamingReporterBase::testGroupEnded(_testGroupStats);
}
void ConsoleReporter::testRunEnded(TestRunStats const& _testRunStats) {
    printTotalsDivider(_testRunStats.totals);
    printTotals(_testRunStats.totals);
    stream « std::endl;
    StreamingReporterBase::testRunEnded(_testRunStats);
}
void ConsoleReporter::testRunStarting(TestRunInfo const& _testInfo) {
    StreamingReporterBase::testRunStarting(_testInfo);
    printTestFilters();
}

void ConsoleReporter::lazyPrint() {

    m_tablePrinter->close();
    lazyPrintWithoutClosingBenchmarkTable();
}

void ConsoleReporter::lazyPrintWithoutClosingBenchmarkTable() {

    if (!currentTestRunInfo.used)
        lazyPrintRunInfo();
    if (!currentGroupInfo.used)
        lazyPrintGroupInfo();

    if (!m_headerPrinted) {
        printTestCaseAndSectionHeader();
        m_headerPrinted = true;
    }
}
void ConsoleReporter::lazyPrintRunInfo() {
    stream « '\n' « getLineOfChars<'~'>() « '\n';
    Colour colour(Colour::SecondaryText);
    stream « currentTestRunInfo->name
        « " is a Catch v" « libraryVersion() « " host application.\n"
        « "Run with -? for options\n\n";

    if (m_config->rngSeed() != 0)
        stream « "Randomness seeded to: " « m_config->rngSeed() « "\n\n";

    currentTestRunInfo.used = true;
}
void ConsoleReporter::lazyPrintGroupInfo() {
    if (!currentGroupInfo->name.empty() && currentGroupInfo->groupsCounts > 1) {
        printClosedHeader("Group: " + currentGroupInfo->name);
        currentGroupInfo.used = true;
    }
}
void ConsoleReporter::printTestCaseAndSectionHeader() {
    assert(!m_sectionStack.empty());
    printOpenHeader(currentTestCaseInfo->name);

    if (m_sectionStack.size() > 1) {
        Colour colourGuard(Colour::Headers);

        auto
```

```
            it = m_sectionStack.begin() + 1, // Skip first section (test case)
            itEnd = m_sectionStack.end();
        for (; it != itEnd; ++it)
            printHeaderString(it->name, 2);
    }

    SourceLineInfo lineInfo = m_sectionStack.back().lineInfo;

    stream « getLineOfChars<'-'>() « '\n';
    Colour colourGuard(Colour::FileName);
    stream « lineInfo « '\n';
    stream « getLineOfChars<'.'>() « '\n' « std::endl;
}

void ConsoleReporter::printClosedHeader(std::string const& _name) {
    printOpenHeader(_name);
    stream « getLineOfChars<'.'>() « '\n';
}
void ConsoleReporter::printOpenHeader(std::string const& _name) {
    stream « getLineOfChars<'-'>() « '\n';
    {
        Colour colourGuard(Colour::Headers);
        printHeaderString(_name);
    }
}

// if string has a : in first line will set indent to follow it on
// subsequent lines
void ConsoleReporter::printHeaderString(std::string const& _string, std::size_t indent) {
    std::size_t i = _string.find(": ");
    if (i != std::string::npos)
        i += 2;
    else
        i = 0;
    stream « Column(_string).indent(indent + i).initialIndent(indent) « '\n';
}

struct SummaryColumn {

    SummaryColumn( std::string _label, Colour::Code _colour )
    :   label( std::move( _label ) ),
        colour( _colour ) {}
    SummaryColumn addRow( std::size_t count ) {
        ReusableStringStream rss;
        rss « count;
        std::string row = rss.str();
        for (auto& oldRow : rows) {
            while (oldRow.size() < row.size())
                oldRow = ' ' + oldRow;
            while (oldRow.size() > row.size())
                row = ' ' + row;
        }
        rows.push_back(row);
        return *this;
    }

    std::string label;
    Colour::Code colour;
    std::vector<std::string> rows;

};

void ConsoleReporter::printTotals( Totals const& totals ) {
    if (totals.testCases.total() == 0) {
        stream « Colour(Colour::Warning) « "No tests ran\n";
    } else if (totals.assertions.total() > 0 && totals.testCases.allPassed()) {
        stream « Colour(Colour::ResultSuccess) « "All tests passed";
        stream « " ("
            « pluralise(totals.assertions.passed, "assertion") « " in "
            « pluralise(totals.testCases.passed, "test case") « ')'
            « '\n';
    } else {

        std::vector<SummaryColumn> columns;
        columns.push_back(SummaryColumn("", Colour::None)
                              .addRow(totals.testCases.total())
                              .addRow(totals.assertions.total()));
        columns.push_back(SummaryColumn("passed", Colour::Success)
                              .addRow(totals.testCases.passed)
                              .addRow(totals.assertions.passed));
        columns.push_back(SummaryColumn("failed", Colour::ResultError)
                              .addRow(totals.testCases.failed)
                              .addRow(totals.assertions.failed));
        columns.push_back(SummaryColumn("failed as expected", Colour::ResultExpectedFailure)
                              .addRow(totals.testCases.failedButOk)
                              .addRow(totals.assertions.failedButOk));

        printSummaryRow("test cases", columns, 0);
        printSummaryRow("assertions", columns, 1);
    }
}
void ConsoleReporter::printSummaryRow(std::string const& label, std::vector<SummaryColumn> const&
cols, std::size_t row) {
    for (auto col : cols) {
        std::string value = col.rows[row];
        if (col.label.empty()) {
            stream « label « ": ";
            if (value != "0")
                stream « value;
            else
                stream « Colour(Colour::Warning) « "- none -";
        } else if (value != "0") {
            stream « Colour(Colour::LightGrey) « " | ";
            stream « Colour(col.colour)
                « value « ' ' « col.label;
        }
    }
    stream « '\n';
}

void ConsoleReporter::printTotalsDivider(Totals const& totals) {
    if (totals.testCases.total() > 0) {
        std::size_t failedRatio = makeRatio(totals.testCases.failed, totals.testCases.total());
        std::size_t failedButOkRatio = makeRatio(totals.testCases.failedButOk,
totals.testCases.total());
        std::size_t passedRatio = makeRatio(totals.testCases.passed, totals.testCases.total());
        while (failedRatio + failedButOkRatio + passedRatio < CATCH_CONFIG_CONSOLE_WIDTH - 1)
            findMax(failedRatio, failedButOkRatio, passedRatio)++;
        while (failedRatio + failedButOkRatio + passedRatio > CATCH_CONFIG_CONSOLE_WIDTH - 1)
            findMax(failedRatio, failedButOkRatio, passedRatio)--;

        stream « Colour(Colour::Error) « std::string(failedRatio, '=');
        stream « Colour(Colour::ResultExpectedFailure) « std::string(failedButOkRatio, '=');
        if (totals.testCases.allPassed())
            stream « Colour(Colour::ResultSuccess) « std::string(passedRatio, '=');
        else
            stream « Colour(Colour::Success) « std::string(passedRatio, '=');
    } else {
        stream « Colour(Colour::Warning) « std::string(CATCH_CONFIG_CONSOLE_WIDTH - 1, '=');
    }
    stream « '\n';
}
void ConsoleReporter::printSummaryDivider() {
    stream « getLineOfChars<'-'>() « '\n';
}

void ConsoleReporter::printTestFilters() {
    if (m_config->testSpec().hasFilters()) {
        Colour guard(Colour::BrightYellow);
        stream « "Filters: " « serializeFilters(m_config->getTestsOrTags()) « '\n';
    }
}

CATCH_REGISTER_REPORTER("console", ConsoleReporter)

} // end namespace Catch

#if defined(_MSC_VER)
#pragma warning(pop)
#endif

#if defined(__clang__)
#  pragma clang diagnostic pop
#endif
// end catch_reporter_console.cpp
// start catch_reporter_junit.cpp

#include <cassert>
#include <sstream>
#include <ctime>
#include <algorithm>
#include <iomanip>

namespace Catch {

    namespace {
        std::string getCurrentTimestamp() {
            // Beware, this is not reentrant because of backward compatibility issues
            // Also, UTC only, again because of backward compatibility (%z is C++11)
            time_t rawtime;
            std::time(&rawtime);
            auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");

#ifdef _MSC_VER
```

```
            std::tm timeInfo = {};
            gmtime_s(&timeInfo, &rawtime);
#else
            std::tm* timeInfo;
            timeInfo = std::gmtime(&rawtime);
#endif

            char timeStamp[timeStampSize];
            const char * const fmt = "%Y-%m-%dT%H:%M:%SZ";

#ifdef _MSC_VER
            std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
#else
            std::strftime(timeStamp, timeStampSize, fmt, timeInfo);
#endif
            return std::string(timeStamp, timeStampSize-1);
        }

        std::string fileNameTag(const std::vector<std::string> &tags) {
            auto it = std::find_if(begin(tags),
                                   end(tags),
                                   [] (std::string const& tag) {return tag.front() == '#'; });
            if (it != tags.end())
                return it->substr(1);
            return std::string();
        }

        // Formats the duration in seconds to 3 decimal places.
        // This is done because some genius defined Maven Surefire schema
        // in a way that only accepts 3 decimal places, and tools like
        // Jenkins use that schema for validation JUnit reporter output.
        std::string formatDuration( double seconds ) {
            ReusableStringStream rss;
            rss « std::fixed « std::setprecision( 3 ) « seconds;
            return rss.str();
        }

    } // anonymous namespace

    JunitReporter::JunitReporter( ReporterConfig const& _config )
        :   CumulativeReporterBase( _config ),
            xml( _config.stream() )
        {
            m_reporterPrefs.shouldRedirectStdOut = true;
            m_reporterPrefs.shouldReportAllAssertions = true;
        }

    JunitReporter::~JunitReporter() {}

    std::string JunitReporter::getDescription() {
        return "Reports test results in an XML format that looks like Ant's junitreport target";
    }

    void JunitReporter::noMatchingTestCases( std::string const& /*spec*/ ) {}

    void JunitReporter::testRunStarting( TestRunInfo const& runInfo )  {
        CumulativeReporterBase::testRunStarting( runInfo );
        xml.startElement( "testsuites" );
    }

    void JunitReporter::testGroupStarting( GroupInfo const& groupInfo ) {
        suiteTimer.start();
        stdOutForSuite.clear();
        stdErrForSuite.clear();
        unexpectedExceptions = 0;
        CumulativeReporterBase::testGroupStarting( groupInfo );
    }

    void JunitReporter::testCaseStarting( TestCaseInfo const& testCaseInfo ) {
        m_okToFail = testCaseInfo.okToFail();
    }

    bool JunitReporter::assertionEnded( AssertionStats const& assertionStats ) {
        if( assertionStats.assertionResult.getResultType() == ResultWas::ThrewException && !m_okToFail
)
            unexpectedExceptions++;
        return CumulativeReporterBase::assertionEnded( assertionStats );
    }

    void JunitReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
        stdOutForSuite += testCaseStats.stdOut;
        stdErrForSuite += testCaseStats.stdErr;
        CumulativeReporterBase::testCaseEnded( testCaseStats );
    }

    void JunitReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
        double suiteTime = suiteTimer.getElapsedSeconds();
        CumulativeReporterBase::testGroupEnded( testGroupStats );
        writeGroup( *m_testGroups.back(), suiteTime );
    }

    void JunitReporter::testRunEndedCumulative() {
        xml.endElement();
    }

    void JunitReporter::writeGroup( TestGroupNode const& groupNode, double suiteTime ) {
        XmlWriter::ScopedElement e = xml.scopedElement( "testsuite" );

        TestGroupStats const& stats = groupNode.value;
        xml.writeAttribute( "name", stats.groupInfo.name );
        xml.writeAttribute( "errors", unexpectedExceptions );
        xml.writeAttribute( "failures", stats.totals.assertions.failed-unexpectedExceptions );
        xml.writeAttribute( "tests", stats.totals.assertions.total() );
        xml.writeAttribute( "hostname", "tbd" ); // !TBD
        if( m_config->showDurations() == ShowDurations::Never )
            xml.writeAttribute( "time", "" );
        else
            xml.writeAttribute( "time", formatDuration( suiteTime ) );
        xml.writeAttribute( "timestamp", getCurrentTimestamp() );

        // Write properties if there are any
        if (m_config->hasTestFilters() || m_config->rngSeed() != 0) {
            auto properties = xml.scopedElement("properties");
            if (m_config->hasTestFilters()) {
                xml.scopedElement("property")
                    .writeAttribute("name", "filters")
                    .writeAttribute("value", serializeFilters(m_config->getTestsOrTags()));
            }
            if (m_config->rngSeed() != 0) {
                xml.scopedElement("property")
                    .writeAttribute("name", "random-seed")
                    .writeAttribute("value", m_config->rngSeed());
            }
        }

        // Write test cases
        for( auto const& child : groupNode.children )
            writeTestCase( *child );

        xml.scopedElement( "system-out" ).writeText( trim( stdOutForSuite ), XmlFormatting::Newline );
        xml.scopedElement( "system-err" ).writeText( trim( stdErrForSuite ), XmlFormatting::Newline );
    }

    void JunitReporter::writeTestCase( TestCaseNode const& testCaseNode ) {
        TestCaseStats const& stats = testCaseNode.value;

        // All test cases have exactly one section - which represents the
        // test case itself. That section may have 0-n nested sections
        assert( testCaseNode.children.size() == 1 );
        SectionNode const& rootSection = *testCaseNode.children.front();

        std::string className = stats.testInfo.className;

        if( className.empty() ) {
            className = fileNameTag(stats.testInfo.tags);
            if ( className.empty() )
                className = "global";
        }

        if ( !m_config->name().empty() )
            className = m_config->name() + "." + className;

        writeSection( className, "", rootSection, stats.testInfo.okToFail() );
    }

    void JunitReporter::writeSection( std::string const& className,
                                      std::string const& rootName,
                                      SectionNode const& sectionNode,
                                      bool testOkToFail) {
        std::string name = trim( sectionNode.stats.sectionInfo.name );
        if( !rootName.empty() )
            name = rootName + '/' + name;

        if( !sectionNode.assertions.empty() ||
            !sectionNode.stdOut.empty() ||
            !sectionNode.stdErr.empty() ) {
            XmlWriter::ScopedElement e = xml.scopedElement( "testcase" );
            if( className.empty() ) {
                xml.writeAttribute( "classname", name );
                xml.writeAttribute( "name", "root" );
            }
            else {
                xml.writeAttribute( "classname", className );
                xml.writeAttribute( "name", name );
            }
            xml.writeAttribute( "time", formatDuration( sectionNode.stats.durationInSeconds ) );
            // This is not ideal, but it should be enough to mimic gtest's
            // junit output.
            // Ideally the JUnit reporter would also handle `skipTest`
            // events and write those out appropriately.
            xml.writeAttribute( "status", "run" );

            if (sectionNode.stats.assertions.failedButOk) {
                xml.scopedElement("skipped")
                    .writeAttribute("message", "TEST_CASE tagged with !mayfail");
            }

            writeAssertions( sectionNode );

            if( !sectionNode.stdOut.empty() )
                xml.scopedElement( "system-out" ).writeText( trim( sectionNode.stdOut ),
      XmlFormatting::Newline );
            if( !sectionNode.stdErr.empty() )
                xml.scopedElement( "system-err" ).writeText( trim( sectionNode.stdErr ),
      XmlFormatting::Newline );
        }
        for( auto const& childNode : sectionNode.childSections )
            if( className.empty() )
                writeSection( name, "", *childNode, testOkToFail );
            else
                writeSection( className, name, *childNode, testOkToFail );
    }

    void JunitReporter::writeAssertions( SectionNode const& sectionNode ) {
        for( auto const& assertion : sectionNode.assertions )
            writeAssertion( assertion );
    }

    void JunitReporter::writeAssertion( AssertionStats const& stats ) {
        AssertionResult const& result = stats.assertionResult;
        if( !result.isOk() ) {
            std::string elementName;
            switch( result.getResultType() ) {
                case ResultWas::ThrewException:
                case ResultWas::FatalErrorCondition:
                    elementName = "error";
                    break;
                case ResultWas::ExplicitFailure:
                case ResultWas::ExpressionFailed:
                case ResultWas::DidntThrowException:
                    elementName = "failure";
                    break;

                // We should never see these here:
                case ResultWas::Info:
                case ResultWas::Warning:
                case ResultWas::Ok:
                case ResultWas::Unknown:
                case ResultWas::FailureBit:
                case ResultWas::Exception:
                    elementName = "internalError";
                    break;
            }

            XmlWriter::ScopedElement e = xml.scopedElement( elementName );

            xml.writeAttribute( "message", result.getExpression() );
            xml.writeAttribute( "type", result.getTestMacroName() );

            ReusableStringStream rss;
            if (stats.totals.assertions.total() > 0) {
                rss « "FAILED" « ":\n";
                if (result.hasExpression()) {
                    rss « "  ";
                    rss « result.getExpressionInMacro();
                    rss « '\n';
                }
                if (result.hasExpandedExpression()) {
                    rss « "with expansion:\n";
                    rss « Column(result.getExpandedExpression()).indent(2) « '\n';
                }
            } else {
                rss « '\n';
            }

            if( !result.getMessage().empty() )
                rss « result.getMessage() « '\n';
            for( auto const& msg : stats.infoMessages )
                if( msg.type == ResultWas::Info )
                    rss « msg.message « '\n';

            rss « "at " « result.getSourceInfo();
            xml.writeText( rss.str(), XmlFormatting::Newline );
        }
    }

    CATCH_REGISTER_REPORTER( "junit", JunitReporter )

} // end namespace Catch
// end catch_reporter_junit.cpp
// start catch_reporter_listening.cpp

#include <cassert>

namespace Catch {

    ListeningReporter::ListeningReporter() {
        // We will assume that listeners will always want all assertions
        m_preferences.shouldReportAllAssertions = true;
    }

    void ListeningReporter::addListener( IStreamingReporterPtr&& listener ) {
        m_listeners.push_back( std::move( listener ) );
    }

    void ListeningReporter::addReporter(IStreamingReporterPtr&& reporter) {
        assert(!m_reporter && "Listening reporter can wrap only 1 real reporter");
        m_reporter = std::move( reporter );
        m_preferences.shouldRedirectStdOut = m_reporter->getPreferences().shouldRedirectStdOut;
    }

    ReporterPreferences ListeningReporter::getPreferences() const {
        return m_preferences;
    }

    std::set<Verbosity> ListeningReporter::getSupportedVerbosities() {
        return std::set<Verbosity>{ };
    }

    void ListeningReporter::noMatchingTestCases( std::string const& spec ) {
        for ( auto const& listener : m_listeners ) {
            listener->noMatchingTestCases( spec );
        }
        m_reporter->noMatchingTestCases( spec );
    }

    void ListeningReporter::reportInvalidArguments(std::string const&arg){
        for ( auto const& listener : m_listeners ) {
            listener->reportInvalidArguments( arg );
        }
        m_reporter->reportInvalidArguments( arg );
    }

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
    void ListeningReporter::benchmarkPreparing( std::string const& name ) {
        for (auto const& listener : m_listeners) {
            listener->benchmarkPreparing(name);
        }
        m_reporter->benchmarkPreparing(name);
    }
    void ListeningReporter::benchmarkStarting( BenchmarkInfo const& benchmarkInfo ) {
        for ( auto const& listener : m_listeners ) {
            listener->benchmarkStarting( benchmarkInfo );
        }
        m_reporter->benchmarkStarting( benchmarkInfo );
    }
    void ListeningReporter::benchmarkEnded( BenchmarkStats<> const& benchmarkStats ) {
        for ( auto const& listener : m_listeners ) {
            listener->benchmarkEnded( benchmarkStats );
        }
        m_reporter->benchmarkEnded( benchmarkStats );
    }

    void ListeningReporter::benchmarkFailed( std::string const& error ) {
        for (auto const& listener : m_listeners) {
            listener->benchmarkFailed(error);
        }
        m_reporter->benchmarkFailed(error);
    }
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

    void ListeningReporter::testRunStarting( TestRunInfo const& testRunInfo ) {
        for ( auto const& listener : m_listeners ) {
            listener->testRunStarting( testRunInfo );
        }
        m_reporter->testRunStarting( testRunInfo );
    }

    void ListeningReporter::testGroupStarting( GroupInfo const& groupInfo ) {
        for ( auto const& listener : m_listeners ) {
            listener->testGroupStarting( groupInfo );
        }
        m_reporter->testGroupStarting( groupInfo );
    }

    void ListeningReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
        for ( auto const& listener : m_listeners ) {
            listener->testCaseStarting( testInfo );
        }
        m_reporter->testCaseStarting( testInfo );
    }

    void ListeningReporter::sectionStarting( SectionInfo const& sectionInfo ) {
        for ( auto const& listener : m_listeners ) {
            listener->sectionStarting( sectionInfo );
        }
        m_reporter->sectionStarting( sectionInfo );
    }

    void ListeningReporter::assertionStarting( AssertionInfo const& assertionInfo ) {
        for ( auto const& listener : m_listeners ) {
            listener->assertionStarting( assertionInfo );
        }
        m_reporter->assertionStarting( assertionInfo );
    }

    // The return value indicates if the messages buffer should be cleared:
    bool ListeningReporter::assertionEnded( AssertionStats const& assertionStats ) {
        for( auto const& listener : m_listeners ) {
            static_cast<void>( listener->assertionEnded( assertionStats ) );
        }
        return m_reporter->assertionEnded( assertionStats );
    }

    void ListeningReporter::sectionEnded( SectionStats const& sectionStats ) {
        for ( auto const& listener : m_listeners ) {
            listener->sectionEnded( sectionStats );
        }
        m_reporter->sectionEnded( sectionStats );
    }

    void ListeningReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
        for ( auto const& listener : m_listeners ) {
            listener->testCaseEnded( testCaseStats );
        }
        m_reporter->testCaseEnded( testCaseStats );
    }

    void ListeningReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
        for ( auto const& listener : m_listeners ) {
            listener->testGroupEnded( testGroupStats );
        }
        m_reporter->testGroupEnded( testGroupStats );
    }

    void ListeningReporter::testRunEnded( TestRunStats const& testRunStats ) {
        for ( auto const& listener : m_listeners ) {
            listener->testRunEnded( testRunStats );
        }
        m_reporter->testRunEnded( testRunStats );
    }

    void ListeningReporter::skipTest( TestCaseInfo const& testInfo ) {
        for ( auto const& listener : m_listeners ) {
            listener->skipTest( testInfo );
        }
        m_reporter->skipTest( testInfo );
    }

    bool ListeningReporter::isMulti() const {
        return true;
    }

} // end namespace Catch
// end catch_reporter_listening.cpp
// start catch_reporter_xml.cpp

#if defined(_MSC_VER)
#pragma warning(push)
#pragma warning(disable:4061) // Not all labels are EXPLICITLY handled in switch
                              // Note that 4062 (not all labels are handled
                              // and default is missing) is enabled
#endif

namespace Catch {
    XmlReporter::XmlReporter( ReporterConfig const& _config )
    :   StreamingReporterBase( _config ),
        m_xml(_config.stream())
    {
        m_reporterPrefs.shouldRedirectStdOut = true;
        m_reporterPrefs.shouldReportAllAssertions = true;
    }

    XmlReporter::~XmlReporter() = default;

    std::string XmlReporter::getDescription() {
        return "Reports test results as an XML document";
    }

    std::string XmlReporter::getStylesheetRef() const {
        return std::string();
    }

    void XmlReporter::writeSourceInfo( SourceLineInfo const& sourceInfo ) {
        m_xml
            .writeAttribute( "filename", sourceInfo.file )
            .writeAttribute( "line", sourceInfo.line );
    }

    void XmlReporter::noMatchingTestCases( std::string const& s ) {
        StreamingReporterBase::noMatchingTestCases( s );
    }

    void XmlReporter::testRunStarting( TestRunInfo const& testInfo ) {
        StreamingReporterBase::testRunStarting( testInfo );
        std::string stylesheetRef = getStylesheetRef();
        if( !stylesheetRef.empty() )
            m_xml.writeStylesheetRef( stylesheetRef );
        m_xml.startElement( "Catch" );
        if( !m_config->name().empty() )
            m_xml.writeAttribute( "name", m_config->name() );
        if (m_config->testSpec().hasFilters())
            m_xml.writeAttribute( "filters", serializeFilters( m_config->getTestsOrTags() ) );
        if( m_config->rngSeed() != 0 )
            m_xml.scopedElement( "Randomness" )
                .writeAttribute( "seed", m_config->rngSeed() );
    }

    void XmlReporter::testGroupStarting( GroupInfo const& groupInfo ) {
        StreamingReporterBase::testGroupStarting( groupInfo );
        m_xml.startElement( "Group" )
            .writeAttribute( "name", groupInfo.name );
    }

    void XmlReporter::testCaseStarting( TestCaseInfo const& testInfo ) {
        StreamingReporterBase::testCaseStarting(testInfo);
        m_xml.startElement( "TestCase" )
            .writeAttribute( "name", trim( testInfo.name ) )
            .writeAttribute( "description", testInfo.description )
            .writeAttribute( "tags", testInfo.tagsAsString() );

        writeSourceInfo( testInfo.lineInfo );

        if ( m_config->showDurations() == ShowDurations::Always )
            m_testCaseTimer.start();
        m_xml.ensureTagClosed();
    }

    void XmlReporter::sectionStarting( SectionInfo const& sectionInfo ) {
        StreamingReporterBase::sectionStarting( sectionInfo );
        if( m_sectionDepth++ > 0 ) {
            m_xml.startElement( "Section" )
                .writeAttribute( "name", trim( sectionInfo.name ) );
            writeSourceInfo( sectionInfo.lineInfo );
            m_xml.ensureTagClosed();
        }
    }

    void XmlReporter::assertionStarting( AssertionInfo const& ) { }

    bool XmlReporter::assertionEnded( AssertionStats const& assertionStats ) {

        AssertionResult const& result = assertionStats.assertionResult;

        bool includeResults = m_config->includeSuccessfulResults() || !result.isOk();

        if( includeResults || result.getResultType() == ResultWas::Warning ) {
            // Print any info messages in <Info> tags.
            for( auto const& msg : assertionStats.infoMessages ) {
                if( msg.type == ResultWas::Info && includeResults ) {
                    m_xml.scopedElement( "Info" )
                            .writeText( msg.message );
                } else if ( msg.type == ResultWas::Warning ) {
                    m_xml.scopedElement( "Warning" )
                            .writeText( msg.message );
                }
            }
        }

        // Drop out if result was successful but we're not printing them.
        if( !includeResults && result.getResultType() != ResultWas::Warning )
            return true;

        // Print the expression if there is one.
        if( result.hasExpression() ) {
            m_xml.startElement( "Expression" )
                .writeAttribute( "success", result.succeeded() )
                .writeAttribute( "type", result.getTestMacroName() );

            writeSourceInfo( result.getSourceInfo() );

            m_xml.scopedElement( "Original" )
                .writeText( result.getExpression() );
            m_xml.scopedElement( "Expanded" )
                .writeText( result.getExpandedExpression() );
        }

        // And... Print a result applicable to each result type.
        switch( result.getResultType() ) {
            case ResultWas::ThrewException:
                m_xml.startElement( "Exception" );
                writeSourceInfo( result.getSourceInfo() );
                m_xml.writeText( result.getMessage() );
                m_xml.endElement();
                break;
            case ResultWas::FatalErrorCondition:
                m_xml.startElement( "FatalErrorCondition" );
                writeSourceInfo( result.getSourceInfo() );
                m_xml.writeText( result.getMessage() );
                m_xml.endElement();
                break;
            case ResultWas::Info:
                m_xml.scopedElement( "Info" )
                    .writeText( result.getMessage() );
                break;
            case ResultWas::Warning:
                // Warning will already have been written
                break;
            case ResultWas::ExplicitFailure:
                m_xml.startElement( "Failure" );
                writeSourceInfo( result.getSourceInfo() );
                m_xml.writeText( result.getMessage() );
                m_xml.endElement();
                break;
            default:
                break;
        }

        if( result.hasExpression() )
            m_xml.endElement();

        return true;
    }

    void XmlReporter::sectionEnded( SectionStats const& sectionStats ) {
        StreamingReporterBase::sectionEnded( sectionStats );
        if( --m_sectionDepth > 0 ) {
            XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResults" );
            e.writeAttribute( "successes", sectionStats.assertions.passed );
            e.writeAttribute( "failures", sectionStats.assertions.failed );
            e.writeAttribute( "expectedFailures", sectionStats.assertions.failedButOk );

            if ( m_config->showDurations() == ShowDurations::Always )
                e.writeAttribute( "durationInSeconds", sectionStats.durationInSeconds );

            m_xml.endElement();
        }
    }

    void XmlReporter::testCaseEnded( TestCaseStats const& testCaseStats ) {
        StreamingReporterBase::testCaseEnded( testCaseStats );
        XmlWriter::ScopedElement e = m_xml.scopedElement( "OverallResult" );
        e.writeAttribute( "success", testCaseStats.totals.assertions.allOk() );

        if ( m_config->showDurations() == ShowDurations::Always )
            e.writeAttribute( "durationInSeconds", m_testCaseTimer.getElapsedSeconds() );

        if( !testCaseStats.stdOut.empty() )
            m_xml.scopedElement( "StdOut" ).writeText( trim( testCaseStats.stdOut ),
XmlFormatting::Newline );
        if( !testCaseStats.stdErr.empty() )
            m_xml.scopedElement( "StdErr" ).writeText( trim( testCaseStats.stdErr ),
XmlFormatting::Newline );

        m_xml.endElement();
    }

    void XmlReporter::testGroupEnded( TestGroupStats const& testGroupStats ) {
        StreamingReporterBase::testGroupEnded( testGroupStats );
        // TODO: Check testGroupStats.aborting and act accordingly.
        m_xml.scopedElement( "OverallResults" )
            .writeAttribute( "successes", testGroupStats.totals.assertions.passed )
            .writeAttribute( "failures", testGroupStats.totals.assertions.failed )
            .writeAttribute( "expectedFailures", testGroupStats.totals.assertions.failedButOk );
        m_xml.scopedElement( "OverallResultsCases")
            .writeAttribute( "successes", testGroupStats.totals.testCases.passed )
            .writeAttribute( "failures", testGroupStats.totals.testCases.failed )
            .writeAttribute( "expectedFailures", testGroupStats.totals.testCases.failedButOk );
        m_xml.endElement();
    }

    void XmlReporter::testRunEnded( TestRunStats const& testRunStats ) {
        StreamingReporterBase::testRunEnded( testRunStats );
        m_xml.scopedElement( "OverallResults" )
            .writeAttribute( "successes", testRunStats.totals.assertions.passed )
            .writeAttribute( "failures", testRunStats.totals.assertions.failed )
            .writeAttribute( "expectedFailures", testRunStats.totals.assertions.failedButOk );
        m_xml.scopedElement( "OverallResultsCases")
            .writeAttribute( "successes", testRunStats.totals.testCases.passed )
            .writeAttribute( "failures", testRunStats.totals.testCases.failed )
            .writeAttribute( "expectedFailures", testRunStats.totals.testCases.failedButOk );
        m_xml.endElement();
    }

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
    void XmlReporter::benchmarkPreparing(std::string const& name) {
        m_xml.startElement("BenchmarkResults")
            .writeAttribute("name", name);
    }

    void XmlReporter::benchmarkStarting(BenchmarkInfo const &info) {
        m_xml.writeAttribute("samples", info.samples)
            .writeAttribute("resamples", info.resamples)
            .writeAttribute("iterations", info.iterations)
            .writeAttribute("clockResolution", info.clockResolution)
            .writeAttribute("estimatedDuration", info.estimatedDuration)
            .writeComment("All values in nano seconds");
    }

    void XmlReporter::benchmarkEnded(BenchmarkStats<> const& benchmarkStats) {
        m_xml.startElement("mean")
            .writeAttribute("value", benchmarkStats.mean.point.count())
            .writeAttribute("lowerBound", benchmarkStats.mean.lower_bound.count())
            .writeAttribute("upperBound", benchmarkStats.mean.upper_bound.count())
            .writeAttribute("ci", benchmarkStats.mean.confidence_interval);
        m_xml.endElement();
        m_xml.startElement("standardDeviation")
            .writeAttribute("value", benchmarkStats.standardDeviation.point.count())
            .writeAttribute("lowerBound", benchmarkStats.standardDeviation.lower_bound.count())
            .writeAttribute("upperBound", benchmarkStats.standardDeviation.upper_bound.count())
            .writeAttribute("ci", benchmarkStats.standardDeviation.confidence_interval);
        m_xml.endElement();
        m_xml.startElement("outliers")
            .writeAttribute("variance", benchmarkStats.outlierVariance)
            .writeAttribute("lowMild", benchmarkStats.outliers.low_mild)
            .writeAttribute("lowSevere", benchmarkStats.outliers.low_severe)
            .writeAttribute("highMild", benchmarkStats.outliers.high_mild)
            .writeAttribute("highSevere", benchmarkStats.outliers.high_severe);
        m_xml.endElement();
        m_xml.endElement();
    }

    void XmlReporter::benchmarkFailed(std::string const &error) {
        m_xml.scopedElement("failed").
            writeAttribute("message", error);
        m_xml.endElement();
    }
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

    CATCH_REGISTER_REPORTER( "xml", XmlReporter )

} // end namespace Catch

#if defined(_MSC_VER)
#pragma warning(pop)
#endif
```

```
// end catch_reporter_xml.cpp

namespace Catch {
    LeakDetector leakDetector;
}

#ifdef __clang__
#pragma clang diagnostic pop
#endif

// end catch_impl.hpp
#endif

#ifdef CATCH_CONFIG_MAIN
// start catch_default_main.hpp

#ifndef __OBJC__

#ifndef CATCH_INTERNAL_CDECL
#ifdef _MSC_VER
#define CATCH_INTERNAL_CDECL __cdecl
#else
#define CATCH_INTERNAL_CDECL
#endif
#endif

#if defined(CATCH_CONFIG_WCHAR) && defined(CATCH_PLATFORM_WINDOWS) && defined(_UNICODE) &&
!defined(DO_NOT_USE_WMAIN)
// Standard C/C++ Win32 Unicode wmain entry point
extern "C" int CATCH_INTERNAL_CDECL wmain (int argc, wchar_t * argv[], wchar_t * []) {
#else
// Standard C/C++ main entry point
int CATCH_INTERNAL_CDECL main (int argc, char * argv[]) {
#endif

    return Catch::Session().run( argc, argv );
}

#else // __OBJC__

// Objective-C entry point
int main (int argc, char * const argv[]) {
#if !CATCH_ARC_ENABLED
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
#endif

    Catch::registerTestMethods();
    int result = Catch::Session().run( argc, (char**)argv );

#if !CATCH_ARC_ENABLED
    [pool drain];
#endif

    return result;
}

#endif // __OBJC__

// end catch_default_main.hpp
#endif

#if !defined(CATCH_CONFIG_IMPL_ONLY)

#ifdef CLARA_CONFIG_MAIN_NOT_DEFINED
#  undef CLARA_CONFIG_MAIN
#endif

#if !defined(CATCH_CONFIG_DISABLE)
// If this config identifier is defined then all CATCH macros are prefixed with CATCH_
#ifdef CATCH_CONFIG_PREFIX_ALL

#define CATCH_REQUIRE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE", Catch::ResultDisposition::Normal,
__VA_ARGS__ )
#define CATCH_REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_REQUIRE_FALSE",
Catch::ResultDisposition::Normal | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )

#define CATCH_REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "CATCH_REQUIRE_THROWS",
Catch::ResultDisposition::Normal, __VA_ARGS__ )
#define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS(
"CATCH_REQUIRE_THROWS_AS", exceptionType, Catch::ResultDisposition::Normal, expr )
#define CATCH_REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES(
"CATCH_REQUIRE_THROWS_WITH", Catch::ResultDisposition::Normal, matcher, expr )
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"CATCH_REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
#endif// CATCH_CONFIG_DISABLE_MATCHERS
#define CATCH_REQUIRE_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_REQUIRE_NOTHROW",
Catch::ResultDisposition::Normal, __VA_ARGS__ )

#define CATCH_CHECK( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CATCH_CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_FALSE",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
#define CATCH_CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CATCH_CHECKED_IF",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CATCH_CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CATCH_CHECKED_ELSE",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CATCH_CHECK_NOFAIL( ... ) INTERNAL_CATCH_TEST( "CATCH_CHECK_NOFAIL",
Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )

#define CATCH_CHECK_THROWS( ... )  INTERNAL_CATCH_THROWS( "CATCH_CHECK_THROWS",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CATCH_CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS(
"CATCH_CHECK_THROWS_AS", exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
#define CATCH_CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES(
"CATCH_CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES(
"CATCH_CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher,
expr )
#endif // CATCH_CONFIG_DISABLE_MATCHERS
#define CATCH_CHECK_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "CATCH_CHECK_NOTHROW",
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )

#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CATCH_CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_CHECK_THAT", matcher,
Catch::ResultDisposition::ContinueOnFailure, arg )

#define CATCH_REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CATCH_REQUIRE_THAT", matcher,
Catch::ResultDisposition::Normal, arg )
#endif // CATCH_CONFIG_DISABLE_MATCHERS

#define CATCH_INFO( msg ) INTERNAL_CATCH_INFO( "CATCH_INFO", msg )
#define CATCH_UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "CATCH_UNSCOPED_INFO", msg )
#define CATCH_WARN( msg ) INTERNAL_CATCH_MSG( "CATCH_WARN", Catch::ResultWas::Warning,
Catch::ResultDisposition::ContinueOnFailure, msg )
#define CATCH_CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer),
"CATCH_CAPTURE",__VA_ARGS__ )

#define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
#define CATCH_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className,
__VA_ARGS__ )
#define CATCH_METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method,
__VA_ARGS__ )
#define CATCH_REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function,
__VA_ARGS__ )
#define CATCH_SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
#define CATCH_DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
#define CATCH_FAIL( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL", Catch::ResultWas::ExplicitFailure,
Catch::ResultDisposition::Normal, __VA_ARGS__ )
#define CATCH_FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "CATCH_FAIL_CHECK",
Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CATCH_SUCCEED( ... ) INTERNAL_CATCH_MSG( "CATCH_SUCCEED", Catch::ResultWas::Ok,
Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )

#define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
#define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
#define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__
)
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG(
__VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
#else
#define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ ) )
#define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
#define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
#define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ ) )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
#endif

#if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
#define CATCH_STATIC_REQUIRE( ... )       static_assert(   __VA_ARGS__ ,      #__VA_ARGS__ ); CATCH_SUCCEED( #__VA_ARGS__ )
#define CATCH_STATIC_REQUIRE_FALSE( ... ) static_assert( !(__VA_ARGS__), "!(" #__VA_ARGS__ ")" ); CATCH_SUCCEED( #__VA_ARGS__ )
#else
#define CATCH_STATIC_REQUIRE( ... )       CATCH_REQUIRE( __VA_ARGS__ )
#define CATCH_STATIC_REQUIRE_FALSE( ... ) CATCH_REQUIRE_FALSE( __VA_ARGS__ )
#endif

// "BDD-style" convenience wrappers
#define CATCH_SCENARIO( ... ) CATCH_TEST_CASE( "Scenario: " __VA_ARGS__ )
#define CATCH_SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario: " __VA_ARGS__ )
#define CATCH_GIVEN( desc )     INTERNAL_CATCH_DYNAMIC_SECTION( "    Given: " « desc )
#define CATCH_AND_GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
#define CATCH_WHEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "     When: " « desc )
#define CATCH_AND_WHEN( desc )  INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
#define CATCH_THEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "     Then: " « desc )
#define CATCH_AND_THEN( desc )  INTERNAL_CATCH_DYNAMIC_SECTION( "      And: " « desc )

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
#define CATCH_BENCHMARK(...) \
    INTERNAL_CATCH_BENCHMARK(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), INTERNAL_CATCH_GET_1_ARG(__VA_ARGS__„), INTERNAL_CATCH_GET_2_ARG(__VA_ARGS__„))
#define CATCH_BENCHMARK_ADVANCED(name) \
    INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

// If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
#else

#define REQUIRE( ... ) INTERNAL_CATCH_TEST( "REQUIRE", Catch::ResultDisposition::Normal, __VA_ARGS__ )
#define REQUIRE_FALSE( ... ) INTERNAL_CATCH_TEST( "REQUIRE_FALSE", Catch::ResultDisposition::Normal | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )

#define REQUIRE_THROWS( ... ) INTERNAL_CATCH_THROWS( "REQUIRE_THROWS", Catch::ResultDisposition::Normal, __VA_ARGS__ )
#define REQUIRE_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "REQUIRE_THROWS_AS", exceptionType, Catch::ResultDisposition::Normal, expr )
#define REQUIRE_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "REQUIRE_THROWS_WITH", Catch::ResultDisposition::Normal, matcher, expr )
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES( "REQUIRE_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::Normal, matcher, expr )
#endif // CATCH_CONFIG_DISABLE_MATCHERS
#define REQUIRE_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "REQUIRE_NOTHROW", Catch::ResultDisposition::Normal, __VA_ARGS__ )

#define CHECK( ... ) INTERNAL_CATCH_TEST( "CHECK", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CHECK_FALSE( ... ) INTERNAL_CATCH_TEST( "CHECK_FALSE", Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::FalseTest, __VA_ARGS__ )
#define CHECKED_IF( ... ) INTERNAL_CATCH_IF( "CHECKED_IF", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CHECKED_ELSE( ... ) INTERNAL_CATCH_ELSE( "CHECKED_ELSE", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CHECK_NOFAIL( ... ) INTERNAL_CATCH_TEST( "CHECK_NOFAIL", Catch::ResultDisposition::ContinueOnFailure | Catch::ResultDisposition::SuppressFail, __VA_ARGS__ )

#define CHECK_THROWS( ... )  INTERNAL_CATCH_THROWS( "CHECK_THROWS", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define CHECK_THROWS_AS( expr, exceptionType ) INTERNAL_CATCH_THROWS_AS( "CHECK_THROWS_AS", exceptionType, Catch::ResultDisposition::ContinueOnFailure, expr )
#define CHECK_THROWS_WITH( expr, matcher ) INTERNAL_CATCH_THROWS_STR_MATCHES( "CHECK_THROWS_WITH", Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) INTERNAL_CATCH_THROWS_MATCHES( "CHECK_THROWS_MATCHES", exceptionType, Catch::ResultDisposition::ContinueOnFailure, matcher, expr )
#endif // CATCH_CONFIG_DISABLE_MATCHERS
#define CHECK_NOTHROW( ... ) INTERNAL_CATCH_NO_THROW( "CHECK_NOTHROW", Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )

#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CHECK_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "CHECK_THAT", matcher, Catch::ResultDisposition::ContinueOnFailure, arg )

#define REQUIRE_THAT( arg, matcher ) INTERNAL_CHECK_THAT( "REQUIRE_THAT", matcher, Catch::ResultDisposition::Normal, arg )
```

```
#endif // CATCH_CONFIG_DISABLE_MATCHERS

#define INFO( msg ) INTERNAL_CATCH_INFO( "INFO", msg )
#define UNSCOPED_INFO( msg ) INTERNAL_CATCH_UNSCOPED_INFO( "UNSCOPED_INFO", msg )
#define WARN( msg ) INTERNAL_CATCH_MSG( "WARN", Catch::ResultWas::Warning, Catch::ResultDisposition::ContinueOnFailure, msg )
#define CAPTURE( ... ) INTERNAL_CATCH_CAPTURE( INTERNAL_CATCH_UNIQUE_NAME(capturer), "CAPTURE",__VA_ARGS__ )

#define TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE( __VA_ARGS__ )
#define TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, __VA_ARGS__ )
#define METHOD_AS_TEST_CASE( method, ... ) INTERNAL_CATCH_METHOD_AS_TEST_CASE( method, __VA_ARGS__ )
#define REGISTER_TEST_CASE( Function, ... ) INTERNAL_CATCH_REGISTER_TESTCASE( Function, __VA_ARGS__ )
#define SECTION( ... ) INTERNAL_CATCH_SECTION( __VA_ARGS__ )
#define DYNAMIC_SECTION( ... ) INTERNAL_CATCH_DYNAMIC_SECTION( __VA_ARGS__ )
#define FAIL( ... ) INTERNAL_CATCH_MSG( "FAIL", Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::Normal, __VA_ARGS__ )
#define FAIL_CHECK( ... ) INTERNAL_CATCH_MSG( "FAIL_CHECK", Catch::ResultWas::ExplicitFailure, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define SUCCEED( ... ) INTERNAL_CATCH_MSG( "SUCCEED", Catch::ResultWas::Ok, Catch::ResultDisposition::ContinueOnFailure, __VA_ARGS__ )
#define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE()

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ )
#define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ )
#define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ )
#define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE(__VA_ARGS__)
#define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ )
#else
#define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ ) )
#define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG( __VA_ARGS__ ) )
#define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
#define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
#define TEMPLATE_PRODUCT_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE( __VA_ARGS__ ) )
#define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( __VA_ARGS__ ) )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, __VA_ARGS__ ) )
#define TEMPLATE_LIST_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE( __VA_ARGS__ ) )
#define TEMPLATE_LIST_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_LIST_TEST_CASE_METHOD( className, __VA_ARGS__ ) )
#endif

#if !defined(CATCH_CONFIG_RUNTIME_STATIC_REQUIRE)
#define STATIC_REQUIRE( ... )       static_assert(   __VA_ARGS__,  #__VA_ARGS__ ); SUCCEED( #__VA_ARGS__ )
#define STATIC_REQUIRE_FALSE( ... ) static_assert( !(__VA_ARGS__), "!(" #__VA_ARGS__ ")" ); SUCCEED( "!(" #__VA_ARGS__ ")" )
#else
#define STATIC_REQUIRE( ... )       REQUIRE( __VA_ARGS__ )
#define STATIC_REQUIRE_FALSE( ... ) REQUIRE_FALSE( __VA_ARGS__ )
#endif

#endif

#define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION( signature )

// "BDD-style" convenience wrappers
#define SCENARIO( ... ) TEST_CASE( "Scenario: " __VA_ARGS__ )
#define SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TEST_CASE_METHOD( className, "Scenario: " __VA_ARGS__ )

#define GIVEN( desc )     INTERNAL_CATCH_DYNAMIC_SECTION( "    Given: " « desc )
#define AND_GIVEN( desc ) INTERNAL_CATCH_DYNAMIC_SECTION( "And given: " « desc )
#define WHEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "     When: " « desc )
#define AND_WHEN( desc )  INTERNAL_CATCH_DYNAMIC_SECTION( " And when: " « desc )
#define THEN( desc )      INTERNAL_CATCH_DYNAMIC_SECTION( "     Then: " « desc )
```

```
#define AND_THEN( desc )  INTERNAL_CATCH_DYNAMIC_SECTION( "      And: " « desc )

#if defined(CATCH_CONFIG_ENABLE_BENCHMARKING)
#define BENCHMARK(...) \
    INTERNAL_CATCH_BENCHMARK(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_),
INTERNAL_CATCH_GET_1_ARG(__VA_ARGS__,,), INTERNAL_CATCH_GET_2_ARG(__VA_ARGS__,,))
#define BENCHMARK_ADVANCED(name) \
    INTERNAL_CATCH_BENCHMARK_ADVANCED(INTERNAL_CATCH_UNIQUE_NAME(C_A_T_C_H_B_E_N_C_H_), name)
#endif // CATCH_CONFIG_ENABLE_BENCHMARKING

using Catch::Detail::Approx;

#else // CATCH_CONFIG_DISABLE

// If this config identifier is defined then all CATCH macros are prefixed with CATCH_
#ifdef CATCH_CONFIG_PREFIX_ALL

#define CATCH_REQUIRE( ... )        (void)(0)
#define CATCH_REQUIRE_FALSE( ... )  (void)(0)

#define CATCH_REQUIRE_THROWS( ... ) (void)(0)
#define CATCH_REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
#define CATCH_REQUIRE_THROWS_WITH( expr, matcher )     (void)(0)
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CATCH_REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
#endif// CATCH_CONFIG_DISABLE_MATCHERS
#define CATCH_REQUIRE_NOTHROW( ... ) (void)(0)

#define CATCH_CHECK( ... )         (void)(0)
#define CATCH_CHECK_FALSE( ... )   (void)(0)
#define CATCH_CHECKED_IF( ... )    if (__VA_ARGS__)
#define CATCH_CHECKED_ELSE( ... )  if (!(__VA_ARGS__))
#define CATCH_CHECK_NOFAIL( ... )  (void)(0)

#define CATCH_CHECK_THROWS( ... )  (void)(0)
#define CATCH_CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
#define CATCH_CHECK_THROWS_WITH( expr, matcher )     (void)(0)
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CATCH_CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
#endif // CATCH_CONFIG_DISABLE_MATCHERS
#define CATCH_CHECK_NOTHROW( ... ) (void)(0)

#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CATCH_CHECK_THAT( arg, matcher )   (void)(0)

#define CATCH_REQUIRE_THAT( arg, matcher ) (void)(0)
#endif // CATCH_CONFIG_DISABLE_MATCHERS

#define CATCH_INFO( msg )          (void)(0)
#define CATCH_UNSCOPED_INFO( msg ) (void)(0)
#define CATCH_WARN( msg )          (void)(0)
#define CATCH_CAPTURE( msg )       (void)(0)

#define CATCH_TEST_CASE( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))
#define CATCH_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
#define CATCH_METHOD_AS_TEST_CASE( method, ... )
#define CATCH_REGISTER_TEST_CASE( Function, ... ) (void)(0)
#define CATCH_SECTION( ... )
#define CATCH_DYNAMIC_SECTION( ... )
#define CATCH_FAIL( ... ) (void)(0)
#define CATCH_FAIL_CHECK( ... ) (void)(0)
#define CATCH_SUCCEED( ... ) (void)(0)

#define CATCH_ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ))

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__)
#define CATCH_TEMPLATE_TEST_CASE_SIG( ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__)
#define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)
#define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD(
className, __VA_ARGS__ )
#else
#define CATCH_TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__) )
#define CATCH_TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__) )
```

```
#define CATCH_TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
#define CATCH_TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS( INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) CATCH_TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ )
#define CATCH_TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) CATCH_TEMPLATE_TEST_CASE_METHOD( className, __VA_ARGS__ )
#endif

// "BDD-style" convenience wrappers
#define CATCH_SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
#define CATCH_SCENARIO_METHOD( className, ... ) INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ), className )
#define CATCH_GIVEN( desc )
#define CATCH_AND_GIVEN( desc )
#define CATCH_WHEN( desc )
#define CATCH_AND_WHEN( desc )
#define CATCH_THEN( desc )
#define CATCH_AND_THEN( desc )

#define CATCH_STATIC_REQUIRE( ... )       (void)(0)
#define CATCH_STATIC_REQUIRE_FALSE( ... ) (void)(0)

// If CATCH_CONFIG_PREFIX_ALL is not defined then the CATCH_ prefix is not required
#else

#define REQUIRE( ... )       (void)(0)
#define REQUIRE_FALSE( ... ) (void)(0)

#define REQUIRE_THROWS( ... ) (void)(0)
#define REQUIRE_THROWS_AS( expr, exceptionType ) (void)(0)
#define REQUIRE_THROWS_WITH( expr, matcher ) (void)(0)
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define REQUIRE_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
#endif // CATCH_CONFIG_DISABLE_MATCHERS
#define REQUIRE_NOTHROW( ... ) (void)(0)

#define CHECK( ... ) (void)(0)
#define CHECK_FALSE( ... ) (void)(0)
#define CHECKED_IF( ... ) if (__VA_ARGS__)
#define CHECKED_ELSE( ... ) if (!(__VA_ARGS__))
#define CHECK_NOFAIL( ... ) (void)(0)

#define CHECK_THROWS( ... )  (void)(0)
#define CHECK_THROWS_AS( expr, exceptionType ) (void)(0)
#define CHECK_THROWS_WITH( expr, matcher ) (void)(0)
#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CHECK_THROWS_MATCHES( expr, exceptionType, matcher ) (void)(0)
#endif // CATCH_CONFIG_DISABLE_MATCHERS
#define CHECK_NOTHROW( ... ) (void)(0)

#if !defined(CATCH_CONFIG_DISABLE_MATCHERS)
#define CHECK_THAT( arg, matcher ) (void)(0)

#define REQUIRE_THAT( arg, matcher ) (void)(0)
#endif // CATCH_CONFIG_DISABLE_MATCHERS

#define INFO( msg ) (void)(0)
#define UNSCOPED_INFO( msg ) (void)(0)
#define WARN( msg ) (void)(0)
#define CAPTURE( ... ) (void)(0)

#define TEST_CASE( ... )  INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
#define TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))
#define METHOD_AS_TEST_CASE( method, ... )
#define REGISTER_TEST_CASE( Function, ... ) (void)(0)
#define SECTION( ... )
#define DYNAMIC_SECTION( ... )
#define FAIL( ... ) (void)(0)
#define FAIL_CHECK( ... ) (void)(0)
#define SUCCEED( ... ) (void)(0)
#define ANON_TEST_CASE() INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ))

#ifndef CATCH_CONFIG_TRADITIONAL_MSVC_PREPROCESSOR
#define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__)
#define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__)
#define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__)
```

```
#define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... )
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
#else
#define TEMPLATE_TEST_CASE( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_NO_REGISTRATION(__VA_ARGS__) )
#define TEMPLATE_TEST_CASE_SIG( ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_SIG_NO_REGISTRATION(__VA_ARGS__) )
#define TEMPLATE_TEST_CASE_METHOD( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_NO_REGISTRATION(className, __VA_ARGS__ ) )
#define TEMPLATE_TEST_CASE_METHOD_SIG( className, ... ) INTERNAL_CATCH_EXPAND_VARGS(
INTERNAL_CATCH_TEMPLATE_TEST_CASE_METHOD_SIG_NO_REGISTRATION(className, __VA_ARGS__ ) )
#define TEMPLATE_PRODUCT_TEST_CASE( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_SIG( ... ) TEMPLATE_TEST_CASE( __VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
#define TEMPLATE_PRODUCT_TEST_CASE_METHOD_SIG( className, ... ) TEMPLATE_TEST_CASE_METHOD( className,
__VA_ARGS__ )
#endif

#define STATIC_REQUIRE( ... )       (void)(0)
#define STATIC_REQUIRE_FALSE( ... ) (void)(0)

#endif

#define CATCH_TRANSLATE_EXCEPTION( signature ) INTERNAL_CATCH_TRANSLATE_EXCEPTION_NO_REG(
INTERNAL_CATCH_UNIQUE_NAME( catch_internal_ExceptionTranslator ), signature )

// "BDD-style" convenience wrappers
#define SCENARIO( ... ) INTERNAL_CATCH_TESTCASE_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME(
C_A_T_C_H_T_E_S_T_ ) )
#define SCENARIO_METHOD( className, ... )
INTERNAL_CATCH_TESTCASE_METHOD_NO_REGISTRATION(INTERNAL_CATCH_UNIQUE_NAME( C_A_T_C_H_T_E_S_T_ ),
className )

#define GIVEN( desc )
#define AND_GIVEN( desc )
#define WHEN( desc )
#define AND_WHEN( desc )
#define THEN( desc )
#define AND_THEN( desc )

using Catch::Detail::Approx;

#endif

#endif // ! CATCH_CONFIG_IMPL_ONLY

// start catch_reenable_warnings.h


#ifdef __clang__
#    ifdef __ICC // icpc defines the __clang__ macro
#        pragma warning(pop)
#    else
#        pragma clang diagnostic pop
#    endif
#elif defined __GNUC__
#    pragma GCC diagnostic pop
#endif

// end catch_reenable_warnings.h
// end catch.hpp
#endif // TWOBLUECUBES_SINGLE_INCLUDE_CATCH_HPP_INCLUDED
```

5.2 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/cmdline.h

```
//
// Created by Lindsay Haslam on 1/11/24.
//

#ifndef EXPRESSIONCLASSES_CMDLINE_H
#define EXPRESSIONCLASSES_CMDLINE_H

#include "catch.h"
#include <cstring>
#include <iostream>

```

```
//class cmdline: public main {
//
//};

void use_arguments(int argc, char **argv);


#endif //HW1_CMDLINE_H
```

5.3 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.cpp File Reference

This header file declares the Expr class hierarchy for representing arithmetic expressions. It includes the abstract base class Expr and its derived classes Num, Var, Add, and Mult. Each class implements functionalities for equality comparison, interpretation (evaluation), variable presence checking, substitution, pretty printing, and standard printing of expressions. This architecture allows for the representation and manipulation of complex arithmetic expressions involving numbers, variables, and the operations of addition and multiplication.

```
#include "Expr.h"
```

5.3.1 Detailed Description

This header file declares the Expr class hierarchy for representing arithmetic expressions. It includes the abstract base class Expr and its derived classes Num, Var, Add, and Mult. Each class implements functionalities for equality comparison, interpretation (evaluation), variable presence checking, substitution, pretty printing, and standard printing of expressions. This architecture allows for the representation and manipulation of complex arithmetic expressions involving numbers, variables, and the operations of addition and multiplication.

Author

Lindsay Haslam

Date

1/18/24

5.4 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h File Reference

This header file defines a hierarchy of expression classes for representing and manipulating arithmetic expressions.

```
#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <stdexcept>
#include <sstream>
```

Classes

- class Expr
- class Num
- class Var
- class Add
- class Mult

Enumerations

- enum **precedence_t** { **prec_none** , **prec_add** , **prec_mult** }

5.4.1 Detailed Description

This header file defines a hierarchy of expression classes for representing and manipulating arithmetic expressions.

Author

Lindsay Haslam

Date

1/18/24

The Expr class hierarchy includes classes for numerical values (Num), variables (Var), addition operations (Add), and multiplication operations (Mult). Each class provides methods for equality checks, interpretation (evaluation), variable presence checks, substitution, and printing. The design supports the construction and manipulation of arithmetic expressions involving integers, variables, addition, and multiplication, allowing for pretty-printing and evaluation with variable substitution.

5.5 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr.h

Go to the documentation of this file.

```

#ifndef EXPRESSIONCLASSES_EXPR_H
#define EXPRESSIONCLASSES_EXPR_H

#include <stdlib.h>
#include <stdio.h>
#include <string>
#include <stdexcept>
#include <sstream>

using namespace std;

typedef enum {
    prec_none,      // = 0
    prec_add,       // = 1
    prec_mult       // = 2
} precedence_t;

class Expr {
public:
    virtual bool equals(Expr *e) = 0;
    virtual int interp() = 0;
    virtual bool has_variable()  = 0;
    virtual Expr* subst(string varName, Expr* replacement)  = 0;
    virtual void print (ostream& os) = 0;

    string to_string();
    void pretty_print(ostream &ostream);
    virtual void pretty_print_at(ostream &ot, precedence_t prec);
    string to_pretty_string();
};

class Num : public Expr {
public:
    int val;
    Num(int val);
    bool equals(Expr *e);
    //Return the value
    int interp();
    //Num will never have a variable.
    bool has_variable();
    Expr* subst( string varName, Expr* replacement);
    virtual void print (ostream& os);
    string to_string();
};

class Var : public Expr{
public:
    string name;
    Var(string name);
    bool equals(Expr *e);
    int interp();
    //Will have a variable.
    bool has_variable();
    Expr* subst( string varName, Expr* replacement);
    virtual void print (ostream& os);
};

class Add : public Expr {
public:
    Expr *lhs, *rhs;
    Add(Expr *lhs, Expr *rhs);
    bool equals(Expr *e);
    //Sum of the subexpression values
    int interp();
    //Check if either have a variable
    bool has_variable();
    Expr* subst( string varName, Expr* replacement);
    virtual void print (ostream& os);
    void pretty_print_at(ostream &ot, precedence_t prec);
};

class Mult : public Expr {
public:
    Expr *lhs, *rhs;
    Mult(Expr *lhs, Expr *rhs);
    bool equals(Expr *e);
    //The product of the subexpression values
    int interp();
    //Check if either have a variable
    bool has_variable();
    Expr* subst(string varName, Expr* replacement);
    virtual void print (ostream& os);
    void pretty_print_at(ostream &ot, precedence_t prec);
};

#endif //EXPRESSIONCLASSES_EXPR_H

```

5.6 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/Expr↩ Tests.cpp File Reference

This test file contains a series of test cases for the Expr class hierarchy, focusing on Var, Num, Add, and Mult classes.

```
#include "ExprTests.h"
```

Functions

- **TEST_CASE** ("Expr Var")

- **TEST_CASE** ("Var Interp Throws")
- **TEST_CASE** ("Var Has Variable")
- **TEST_CASE** ("Var Subst")
- **TEST_CASE** ("Expr Add")
- **TEST_CASE** ("Add Interp")
- **TEST_CASE** ("Add Has Variable")
- **TEST_CASE** ("Add Subst")
- **TEST_CASE** ("Expr Num")
- **TEST_CASE** ("Num Interp")
- **TEST_CASE** ("Num Has Variable")
- **TEST_CASE** ("Expr Mult")
- **TEST_CASE** ("Mult Interp")
- **TEST_CASE** ("Mult Has Variable")
- **TEST_CASE** ("Mult Subst")
- **TEST_CASE** ("Nabil's Tests for Pretty Print")
- **TEST_CASE** ("Pretty Print Mult Expressions")
- **TEST_CASE** ("Pretty Print Add Expressions")
- **TEST_CASE** ("Pretty Print Var Expressions")
- **TEST_CASE** ("Multiple Types of Operations")

5.6.1 Detailed Description

This test file contains a series of test cases for the Expr class hierarchy, focusing on Var, Num, Add, and Mult classes.

Author

Lindsay Haslam

Date

1/18/24

It includes tests for equality checks, interpretation (evaluation), variable presence checks, substitution, and pretty printing functionalities of arithmetic expressions. Each test case is designed to verify the correct behavior of the classes and their interactions, ensuring that expressions are correctly manipulated and evaluated according to the rules of arithmetic and variable substitution.

5.7 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/ExprTests.h

```
//
// Created by Lindsay Haslam on 1/18/24.
//

#ifndef EXPRESSIONCLASSES_EXPRTESTS_H
#define EXPRESSIONCLASSES_EXPRTESTS_H

#include "catch.h"
#include "Expr.h"


#endif //EXPRESSIONCLASSES_EXPRTESTS_H
```

5.8 /Users/lindsayhaslam/CS6015/HW4/ExpressionClasses/main.cpp File Reference

Main entry point for the program. This file contains the main function that serves as the entry point of the program.

```
#include "cmdline.h"
```

Functions

- int **main** (int argc, char **argv)

5.8.1 Detailed Description

Main entry point for the program. This file contains the main function that serves as the entry point of the program.

Author

Lindsay Haslam

Date

2/6/2024